PYTHON CƠ BẢN

NEW

....Rất là CƠ BẢN

Bởi Võ Duy Tuấn

Giới thiệu

Hiện nay, Python là một trong những ngôn ngữ lập trình đang được chú ý bởi tính đa dạng về ứng dụng, thư viện phong phú và cộng đồng đông đảo.

Đã làm việc với PHP 10 năm, và có những tác vụ mà PHP khó mà thực hiện tối ưu được, khiến mình phải tiếp cận với Python trong giai đoạn này.

Cuốn sách nhỏ này được viết trong quá trình mình bắt đầu học Python và giải quyết các bài toán cơ bản theo nhu cầu của mình.

Hy vọng những ghi chép của mình cũng sẽ giúp ích cho những ai đang quan tâm đến việc ứng dụng Python vào công việc và xử lý hiện tại.

Mục lục

Sách được chia làm 15 chương, mỗi chương sẽ trình bày 1 khía cạnh của Python mà mình sẽ gặp phải và sẽ hữu ích

khi biết các kiến thức này trong việc áp dụng Python vào công việc trong tương lai.

1. Hello world
2. Cú pháp
3. Phân chia module
4. Class
5. Thao tác trên tập tin
6. Xử lý hình ảnh
7. Xử lý file JSON
8. Xử lý file XML
9. Kết nối MySQL
10. Kết nối Redis
11. Kết nối Memcached
12. Kết nối RabbitMQ
13. Restful Client
14. Gởi email với SMTP
15. Socket Programming

Tác giả

- Tên: Võ Duy Tuấn
- Email: tuanmaster2012@gmail.com
- Facebook: https://www.facebook.com/voduytuan

Chương 1. Hello world

Python là một ngôn ngữ phiên dịch (Interpreter Language), tức là không cần build thành file thực thi mà chạy trực tiếp như PHP.

Hiện tại Python có 2 nhánh chính là 2.x và 3.x. Ở nhánh 2.x đã dừng phát triển và đang đứng ở phiên bản 2.7. Nhánh Python 3.x thì vẫn đang được tiếp tục phát triển.

Website chính thức của Python: www.python.org

Cài đặt

Python hỗ trợ hầu hết các nền tảng và rất dễ tìm thấy sẵn trên một số hệ điều hành như Mac OS…

Để biết là hệ thống của bạn đã cài Python chưa, có thể vào màn hình command line và gõ:

```
$ python --version
```

Nếu đã cài đặt python thì sẽ hiển thị thông tin phiên bản python. Nếu báo lỗi thì đồng nghĩa với bạn chưa cài đặt Python.

Có thể tham khảo cách cài đặt Python tại: https://www.python.org/downloads/

Công cụ phát triển

Chỉ cần dùng một text editor là bạn có thể viết được code python hoặc có thể dùng các công cụ cao cấp hơn (IDE) như Aptana, PyCharm...Các IDE thường hỗ trợ thêm quá trình phân tích cú pháp dòng lệnh, debug...trong phạm vi cuốn sách nhỏ này thì mình hướng đến cách thực thi python bằng dòng lệnh.

Hello world

Tạo một file có tên là `helloworld.py` và có nội dung như sau:

```
print 'Hello world'
```

`print` là lệnh cơ bản nhất để xuất một biến ra (thường là màn hình)

Sau đó, vào màn hình command line, di chuyển đến thư mục chứa file này và gõ.

```
$ python helloworld.py
```

Nếu thấy xuất hiện dòng chữ `Hello world` tức là bạn đã hoàn thành việc viết ứng dụng python đầu tiên.

Chương 2. Cú pháp

2.1. Biến số

Khai báo biến bằng một câu lệnh gán.

```
a = 1
```

bạn có thể gán nhiều loại giá trị (số, chuỗi) cho một biến.

```
a = 1
a = 'Hello World'
a = [1, 2, 3]
a = [1.2, 'Hello', 'W', 2]
```

2.2. Toán tử số học

Python cũng hỗ trợ một số toán tử toán học thông dụng như:

- `+` phép cộng

- `-` phép trừ
- `*` phép nhân
- `/` phép chia
- `%` phép chia lấy dư (modulo)

2.3. Boolean và Toán tử logic

Giá trị đúng và sai tương ứng là `True` và `False`.

- `not` để đảo giá trị.
- `and` phép tính logic và (AND)
- `or` phép tính logic hoặc (OR)

Một số phép so sánh thông thường như `<` (bé hơn), `<=` (bé hơn hoặc bằng), `>` (lớn hơn), `>=` (lớn hơn hoặc bằng), `==` (bằng), `!=` (khác) để so sánh 2 giá trị.

Hỗ trợ dạng so sánh kép như:

```
x = 2

1 < x < 3   # True
10 < x < 20 # False
3 > x <= 2  # True
2 == x < 4  # True
```

Toán tử kiểm tra phần tử trong một tập hợp: - `in` kiểm tra có tồn tại - `not in` kiểm không tồn tại

```
'good' in 'this is a greate example'             # F
alse
'good' not in 'this is a greate example'    # True
```

2.4. Cấu trúc điều khiển

Python hỗ trợ một số cấu trúc điều khiển thông dụng. Hầu hết các cấu trúc điều khiển đều dựa vào thụt đầu dòng (indention) để tạo thành một block xử lý, thay vì sử dụng `{...}` như các ngôn ngữ khác (PHP, Javascript) ### 2.4.1. If...elif...else

```
if condition1 :
    indentedStatementBlockForTrueCondition1
elif condition2 :
    indentedStatementBlockForFirstTrueCondition2
elif condition3 :
    indentedStatementBlockForFirstTrueCondition3
elif condition4 :
    indentedStatementBlockForFirstTrueCondition4
else:
    indentedStatementBlockForEachConditionFalse
```

2.4.2. Switch…case

Python không có cấu trúc `switch`…`case`

2.4.3. For…in

```
for iterating_var in sequence:
    statements(s)
```

Ví dụ:

```
for letter in 'Python':     # First Example
    print 'Current Letter :', letter

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # Second Example
    print 'Current fruit :', fruit

print "Good bye!"
```

Kết quả hiển thị của ví dụ trên:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

2.4.4. While

```
while expression:
    statement(s)
```

Ví dụ:

```
count = 0
while (count < 9):
   print 'The count is:', count
   count = count + 1

print "Good bye!"
```

Kết quả hiển thị của ví dụ trên:

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

2.5. Hàm

Khai báo hàm theo cú pháp:

```
def functionname(param, param2,..):
    statements(s)
```

Hàm nếu không trả dữ liệu thì mặc định sẽ trả về giá trị `None`

Ví dụ khai báo hàm tính và trả về giá trị tổng của 2 tham số đầu vào:

```
def sum(a, b):
    return (a+b)
```

Cách gọi hàm:

```
sum(1, 2)
(trả về giá trị là 3)
```

Hàm có hỗ trợ giá trị mặc định cho tham số khi không truyền vào. Ví dụ hàm sau:

```
def plus(c, d = 10):
    return (c+d)
```

Nếu gọi hàm trên như sau:

```
plus(2)
(kết quả trả về là 12)
```

Một khác biệt trong cách gọi hàm của Python so với PHP là chúng ta có thể thay đổi thứ tự tham số truyền vào bằng

cách đặt tên tham số khi gọi hàm. Ví dụ ta có thể gọi hàm `sum(a,b)` ở ví dụ trên bằng cách truyền tham số `b` trước `a` như sau:

```
sum(b = 1, a = 10)
```

2.6. Xử lý chuỗi

Một chuỗi có thể khai báo bằng dấu nháy đôi `"` hoặc đơn `'`. Ví dụ các chuỗi sau:

```
str1 = "Hello"
str2 = 'world'
```

Có thể truy xuất từng ký tự trong một chuỗi theo hình thức index, ví dụ: `str1[0]`, `str1[1]`...
Có thể sử dụng 3 dấu nháy (đôi hoặc đơn) để khai báo chuỗi trên nhiều dòng. Ví dụ:

```
paragraph = """This is line 1
This is line 2
This is line 3"""
```

2.6.1. Nối chuỗi

Có thể tạo một chuỗi dài từ việc nối các chuỗi lại theo cú pháp:

```
str = str1 + " " + str2
```

2.6.2. Trích xuất chuỗi con

Có thể tạo các chuỗi con thông qua toán tử lấy khoản `[start:end]` (range). Mặc định `start` là từ vị trí đầu chuỗi (`0`) và `end` là đến vị trí cuối chuỗi. Ví dụ:

```
str = 'Hello world'

print str[0:4]
(Hiển thị "Hell")

print str[:4]
(Hiển thị "Hell")

print str[-3:]
(Hiển thị "rld")

print str[6:-3]
(Hiển thị "wo")
```

2.6.3. Lấy độ dài của chuỗi

Sử dụng hàm `len(...)` để trả về độ dài của chuỗi. Ví dụ:

```
count = len("Hello world")
(count có giá trị 11)
```

2.6.4. Tìm & thay thế nội dung

Có thể tìm và thay thế trong chuỗi bằng cách gọi phương

thức `replace(search, replace[, max])` của một chuỗi. Ví dụ:

```
str = 'Hello world'
newstr = str.replace('Hello', 'Bye')
print newstr
(Sẽ hiển thị chuỗi "Bye world" trên màn hình)
```

2.6.5. Tìm vị trí chuỗi con

Có thể tìm vị trí của một chuỗi con trong chuỗi lớn bằng cách gọi phương thức `find(str, beg=0 end=len(string))`. Bắt đầu là vị trí `0`, nếu không tìm ra thì trả về `-1`. Ví dụ:

```
str = 'Hello world'
print str.find('world')
(hiển thị 6)

print str.find('Bye');
(hiển thị -1)
```

Hàm `find()` sẽ tìm theo thứ tự từ trái qua phải của chuỗi,

tức là từ lần xuất hiện đầu tiên. Có thể dùng hàm `rfind()` để tìm theo vị trí từ cuối chuỗi về phía trước.

2.6.6. Tách chuỗi

Có thể tách chuỗi dựa theo một chuỗi delimeter bằng cách gọi phương thức `split(str="", num=string.count(str))`. Ví dụ:

```
str = 'Hello world'
print str.split(' ')
(Trả về một mảng có 2 phần tử là 2 chuỗi "Hello" và
 "world")
```

Có thể sử dụng hàm `splitlines()` để tách chuỗi theo từng hàng và loại bỏ ký tự NEWLINE.

2.6.7. Trim ký tự khoảng trắng

Có thể loại bỏ các ký tự (mặc định là ký tự khoảng trắng) trước và sau một chuỗi, bằng cách gọi các phương thức sau:

- `strip([chars])`: loại bỏ trước và sau chuỗi
- `lstrip([chars])`: loại bỏ phía trước chuỗi
- `rstrip([chars])`: loại bỏ phía sau chuỗi

2.6.8. Một số hàm xử lý chuỗi

- `isnumeric()`: Kiểm tra một chuỗi có phải là chuỗi số
- `lower()`: Chuyển chuỗi hết thành chữ thường
- `upper()`: Chuyển chuỗi hết thành chữ HOA

2.7. List

List trong Python là cấu trúc mảng và các phần tử có index có thứ tự. Không như PHP, key của một mảng có thể vừa là số, vừa là chuỗi (associated array).

Trong Python, muốn tạo một mảng có key là chuỗi thì sẽ sử dụng cấu trúc Dictionary (phần tiếp tiếp). Trong phần này, chúng ta sẽ nói đến List. Một List được khai báo như mảng trong JSON. Sử dụng `[..]` để khai báo một mảng. Ví dụ:

```
numbers = [1, 2, 3, 4, 5]
names = ['Marry', 'Peter']
```

Có thể truy xuất từng phần tử của mảng bằng index, phần tử đầu tiên có thứ tự là `0` . Ví dụ:

```
print numbers[0]
(Hiển thị 1)

print numbers[-3]
(Hiển thị 3)

print names[1]
(Hiển thị 'Peter')
```

Để biết được số lượng phần tử của 1 List, có thể sử dụng hàm `len(array)` để lấy số lượng phần tử của mảng tham số truyền vào.

2.7.1. Kiểm tra sự tồn tại của một phần tử

2.7.1.1. Kiểm tra theo Index

Trong nhiều trường hợp bạn muốn truy xuất một phần tử bất kỳ (dựa vào index) của mảng thì nếu truy xuất đến một phần tử không tồn tại thì ứng dụng sẽ báo lỗi. Do đó, trước khi truy xuất một phần tử, bạn cần kiểm tra xem phần tử này đã tồn tại hay chưa. Hiện tại python không hỗ trợ hàm nào để kiểm tra sự tồn tại của một phần tử trong mảng.

Có 2 cách thường thấy để kiểm tra đó là “Look before you leap” (LBYL) và “Easier to ask forgiveness than permission” (EAFP).

Ví dụ về “Look before you leap (LBYL)”:

```
if index < len(array):
    array[index]
else:
    # handle this
```

Ví dụ về “Easier to ask forgiveness than permission” (EAFP):

```
try:
    array[index]
except IndexError:
    # handle this
```

2.7.1.2. Kiểm tra theo giá trị

Để kiểm tra một giá trị có tồn tại / không tồn tại trong mảng hay không thì có thể sử dụng toán tử `in` / `not in`. Ví dụ:

```
mylist = ['a', 'b', 'c']

print 'a' in mylist
(Hiển thị True)

print 'b' not in mylist
(Hiển thị False)
```

2.7.2. Trích xuất mảng con

Tương tự như chuỗi, tó thể tạo các mảng con thông qua toán tử lấy khoản `[start:end]` (range). Mặc định `start` là

từ vị trí đầu chuỗi (`0`) và `end` là đến vị trí cuối chuỗi. Ví dụ:

```
numbers = ['a', 'b', 'c', 'd']

print numbers[:2]
(Hiển thị ['a', 'b'])

print numbers[-2:]
(Hiển thị ['c', 'd'])
```

2.7.3. Xóa phần tử của mảng

Có thể xóa một phần tử thông qua toán tử `del`. Thứ tự của các phần tử sẽ dịch chuyển tùy vào vị trí của phần tử bị xóa. Ví dụ:

```
numbers = [1, 2, 3, 4, 5]
del numbers[0]
print numbers
(Hiển thị [2, 3, 4, 5])
```

Bạn có thể xóa một khoản dựa vào toán tử lấy khoản `[start:end]`. Ví dụ:

```
numbers = [1, 2, 3, 4, 5, 6, 7]
del numbers[2:4]
print numbers
(Hiển thị [1, 2, 5, 6, 7])
```

2.7.4. Nối 2 mảng

Bạn có thể sử dụng toán tử `+` để nối giá trị của 2 mảng và tạo ra một mảng lớn có số lượng phần tử là tổng số lượng phần tử của 2 mảng con. Ví dụ:

```
a = [1, 2]
b = [1, 3]

print a + b
(Hiển thị [1, 2, 1, 3])
```

2.7.5. Thêm phần tử vào mảng

Nếu bạn muốn thêm phần tử vào một mảng đã tồn tại, hãy dùng phương thức `list.append(newvalue)` để thêm phần tử có giá trị `newvalue` vào cuối mảng `list`. Ví dụ:

```
numbers = [1, 2, 3]
numbers.append(4)
print numbers
(Hiển thị [1, 2, 3, 4]
```

2.7.6. Lấy phần tử cuối mảng

Nếu muốn lấy phần tử cuối cùng của mảng ra khỏi mảng, có thể sử dụng phương thức `list.pop()`, sẽ trả về giá trị của phần tử cuối cùng và mảng bây giờ sẽ không còn phần tử này.

```
numbers = [1, 2, 3]
mynumber = numbers.pop()
print mynumber
(Hiển thị 3)

print numbers
(Hiển thị [1, 2])
```

2.7.7. Tìm một giá trị trong mảng

Nếu bạn muốn tìm vị trí (index) của một giá trị trong một

mảng, có thể dùng phương thức `list.index(obj)`. Nếu tìm thấy sẽ trả về index của phần tử đầu tiên tìm thấy. Nếu không tìm thấy sẽ quăng Exception. Ví dụ:

```
aList = [123, 'xyz', 'zara', 'abc'];

print "Index for xyz : ", aList.index('xyz')
print "Index for zara : ", aList.index('zara')
```

Khi chạy sẽ hiển thị kết quả:

```
Index for xyz :  1
Index for zara :  2
```

2.7.8. Đảo ngược giá trị của mảng

Để đảo ngược thứ tự các giá trị của một mảng, sử dụng phương thức `list.reverse()`. Phương thức này không trả về kết quả mà thay đổi trực tiếp mảng `list`. Ví dụ:

```
numbers = [1, 2, 3, 4]
numbers.reverse()
print numbers
(Hiển thị [4, 3, 2, 1])
```

2.7.9. Sắp xếp giá trị các phần tử

Để sắp xếp thứ tự của giá trị trong mảng, sử dụng phương thức `list.sort([func])` để sắp xếp. Nếu tham số đầu vào là hàm `func` không truyền vào thì mặc định là sắp xếp theo giá trị tăng dần. Phương thức này không trả về kết quả mà thay đổi trực tiếp mảng `list`. Ví dụ:

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz']
aList.sort()
print "List : ", aList
(Hiển thị List :  [123, 'abc', 'xyz', 'xyz', 'zara'
])
```

Cách triển khai hàm compare `func()` cũng giống như hàm usort trong PHP. Hàm trả về các giá trị `0`, `-1` và `1`.

2.8. Tuple

Tuple cũng là một cấu trúc mảng, tương tự như cấu trúc List. Một số điểm khác nhau cơ bản là khai báo Tuple sử dụng cặp dấu ngoặc `(...)` và một tuple đã được khai báo rồi thì không thay đổi được giá trị (immutable) và không hỗ trợ các phương thức như `append()`, `pop()`...Ví dụ:

```
mytuple = ('x', 'y', 'z')
print mytuple
(Hiển thị ('x', 'y', 'z'))
```

Vẫn hỗ trợ các cách để truy xuất phần tử giống List như là truy xuất theo index, range, tìm kiếm...

2.9. Dictionary

Dictionary cũng là một cấu trúc mảng, nhưng các phần tử bao gồm key và value. Nếu bạn có biết JSON thì cấu trúc Dictionary tương tự như một object json. Một Dictionary được khai báo bằng cặp dấu ngoặc `{...}`. Ví dụ:

```
point = {'x': 1, 'y': 2}
```

Truy xuất một giá trị dựa vào key của đối tượng. Ví dụ:

```
point = {'x': 3, 'y': 6, 'z' : 9}
print point[x]
(Hiển thị 3)
```

2.9.1. Thêm một phần tử

Để thêm một phần tử vào đối tượng đã khai báo, sử dụng cấu trúc `dict[key] = value`. Ví dụ:

```
user = {'name': 'Jone', 'age': 30}
user['country'] = 'Vietnam'
print user
(Hiển thị {'country': 'Vietnam', 'age': 30, 'name':
 'Jone'})
```

2.9.2. Một số hàm, phương thức thông dụng:

- `dict.clear()`: Xóa toàn bộ dữ liệu bên trong đối

tượng

- `dict.copy()`: Trả về một bản copy của đối tượng
- `dict.fromkeys(seq[, value])`: Tạo một đối tượng với danh sách key từ seq và nếu có truyền `value` thì lấy đó làm giá trị cho các phần tử.
- `dict.has_key(key)`: kiểm tra một key có tồn tại trong đối tượng hay không.
- `dict.keys()`: Trả về một List chứa các key
- `dict.values()`: Trả về một List chứa các value

Chương 3. Phân chia module

Tất cả ví dụ cho đến thời điểm này đều được thực thi trong command line hoặc từ một file python `.py`. Tuy nhiên, đối với các ứng dụng lớn, có nhiều chức năng thì phân chia nhỏ dự án thành các file khác nhau sẽ giúp dễ bảo trì và tái sử dụng các thành phần đã thiết kế.

Chương này sẽ giúp bạn thiết kế các tính năng theo mô hình các module và khi cần thì sẽ gọi file tương ứng và sử dụng.

3.1. Các loại module / thư viện

Có 3 loại module thường thấy là:

1. Viết bằng Python: có phần mở rộng là `.py`
2. Các thư viện liên kết động: có phần mở rộng là `.dll`, `.pyd`, `.so`, `.sl`,...
3. C-Module liên kết với trình phiên dịch.

3.2. Đường dẫn tìm để load module

Để tải một module vào script của bạn, sử dụng cú pháp đơn giản:

```
import modulename
```

khi gặp câu lệnh trên thì trình biên dịch sẽ tiến hành tìm kiếm file module tương ứng theo thứ tự thư mục sau:

1. Thư mục hiện hành mà script đang gọi
2. Các thư mục trong PYTHONPATH (nếu có set)
3. Các thư mục cài đặt chuẩn trên Linux/Unix..

Có thể biết được đường dẫn mà một module đã được load bằng đoạn code dưới đây:

```
import math
math.__file__
(Ví dụ trả về '/usr/lib/python2.5/lib-dynload/math.
so')

import random
random.__file__
(Ví dụ trả về '/usr/lib/python2.5/random.pyc')
```

3.3. Lấy danh sách thuộc tính và phương thức của một module

Để lấy được danh sách các thuộc tính và phương thức mà module hỗ trợ, sử dụng hàm `dir(modulename)`. Ví dụ:

```
dir(math)
['__doc__', '__file__', '__name__', '__package__',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degree
s', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'fa
ctorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma'
, 'hypot', 'isinf', 'isnan', 'ldexp', 'lgamma', 'lo
g', 'log10', 'log1p', 'modf', 'pi', 'pow', 'radians
', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

Có thể gọi hàm `dir()` không truyền tham số để lấy các thuộc tính và phương thức của scope hiện tại đang thực thi.

3.4. Cách khai báo và sử dụng module

Giả sử bạn tạo một file python `mymath.py` có nội dung như sau:

```
def cong(a, b):
    return a + b

def tru(a, b):
    return a - b

def nhan(a, b):
    return a * b
```

Sau đó, tạo một file có tên `myexample.py`, trong cùng thư mục với file `mymath.py` vừa tạo ở trên, có nội dung như sau:

```
import mymath

num1 = 1
num2 = 2

print 'Tong hai so la: ', mymath.cong(num1, num2)
```

Vào command line, thực hiện gọi file myexample như sau:

```
$ python myexample.py
```

Sau khi thực hiện sẽ hiển thị lên màn hình là

```
Tong hai so la: 3
```

3.5. Package module

Có thể gom nhiều module `.py` vào một thư mục và tên thư mục là tên của package và tạo một file `__init__.py` trong thư mục này.

Như vậy, cấu trúc thư của một package sẽ như sau:

```
|-- mypack
|    |-- __init__.py
|    |-- mymodule1.py
|    |-- mymodule2.py
|
```

Có thể sử dụng `mymodule1` theo cú pháp import sau:

```
import mypack.mymodule1
```

hoặc

```
import mypack.mymodule1 as mymodule1
```

hoặc

```
import mypack.mymodule1 as mod
```

Khi sử dụng một module thuộc một package thì các lệnh trong file `__init__.py` sẽ được thực hiện trước. Thông thường thì file `__init__.py` sẽ rỗng.

Có thể tạo các subpackage bên trong một package theo đúng cấu trúc thư mục, có file `__init__.py`. Ví dụ:

```
import mypack.mysubpack.mysubsubpack.module
```

Chương 4. Class

Lập trình hướng đối tượng là một khái niệm không thể thiếu trong hầu hết các ngôn ngữ thông dụng hiện nay. Python cũng hỗ trợ lập trình hướng đối tượng với các khái niệm Class, Object, Override…

4.1. Khai báo một Class

Khai báo một class theo cú pháp sau:

```
class myclass([parentclass]):
    assignments
    def __init__(self):
        statements
    def method():
        statements
    def method2():
        statements
```

Ví dụ một class:

```
class animal():
```

```
    name = ''
    age = 0
    def __init__(self, name = '', age = 0):
        self.name = name
        self.age = age
    def show(self):
        print 'My name is ', self.name
    def run(self):
        print 'Animal is running...'
    def go(self):
        print 'Animal is going...'


class dog(animal):
    def run(self):
        print 'Dog is running...'

myanimal = animal()
myanimal.show()
myanimal.run()
myanimal.go()

mydog = dog('Lucy')
mydog.show()
mydog.run()
mydog.go()
```

Sau khi thực thi sẽ cho ra kết quả:

```
My Name is
Animal is running...
Animal is going...
My Name is Lucy
Dog is running...
Animal is going...
```

Trong ví dụ trên thì:

- `animal` và `dog` là 2 class. Trong đó class `dog` kế thừa từ class cha là `animal` nên sẽ có các phương thức của class `animal`.
- `name` và `age` là thuộc tính (Attribute) của class.
- Phương thức `__init__(self)` là hàm tạo của class. Hàm này sẽ được gọi mỗi khi có một object mới được tạo (từ một class), gọi là quá trình tạo instance.
- `show()`, `run()` và `go()` là 2 phương thức của 2 class. Khi khai báo phương thức có kèm tham số `self` dùng để truy cập ngược lại object đang gọi. Lúc gọi phương

thức thì không cần truyền tham số này.

- Phương thức `run()` của class `dog` gọi là `override` của phương thức `run()` của class `animal`.

Chương 5. Thao tác trên tập tin và Thư mục

Nội dung chương này sẽ hướng dẫn các thao tác liên quan đến tập tin và thư mục.

5.1. Tập tin (File)

5.1.1. Mở file

Trước khi muốn đọc hoặc ghi file, bạn cần có thao tác mở file theo cú pháp:

```
fh = open(filepath, mode)
```

Trong đó, `filepath` là đường dẫn của file sẽ mở và `mode` là chế độ để mở. Có một số chế độ là:

- `r`: mở để đọc nội dung (mặc định)
- `w`: mở để ghi nội dung
- `a`: mở để ghi thêm nội dung vào cuối file.

- `r+`: mở để đọc và ghi. Con trỏ nằm ở đầu file.
- `w+`: mở để đọc và ghi. Ghi đè nếu file đã tồn tại, nếu file chưa tồn tại thì tạo file mới để ghi.
- `a+`: mở để đọc và thêm vào cuối file. Con trỏ nằm ở cuối file. Nếu file chưa tồn tại thì tạo file mới để ghi.

Mặc định là mở file text, nếu muốn mở file nhị phân (binary) thì thêm `b`, ví dụ: `rb`, `wb`, `ab`, `rb+`, `wb+`, `ab+`.

Ví dụ:

```
f1 = open('test.txt', 'r')
f2 = open('access_log', 'a+'
```

Sau khi gọi hàm `open()` thành công thì sẽ trả về một object có các thuộc tính:

- `closed`: True nếu file đã đóng
- `mode`: chế độ khi mở file
- `name`: tên của file
- `softspace`: cờ đánh dấu softspace khi dùng với hàm

`print`

5.1.2. Đọc nội dung từ file

Sau khi file đã mở ở chế độ đọc thì gọi phương thức `read([count])` để trả về toàn bộ nội dung của file. Ví dụ:

```
f1 = open('test.txt', 'r')
data = f1.read();
```

Hàm `read()` có nhận một tham số là số lượng byte muốn đọc. Nếu không truyền vào thì sẽ đọc hết nội dung của file. Ví dụ:

```
f2 = open('log.txt', 'r')
buffdata = f2.read(1024)
```

5.1.3. Ghi nội dung vào file

Nếu file được mở ở chế độ có thể ghi thì có thể dùng phương thức `write()` để ghi một nội dung vào file. Ví dụ:

```
f2 = open('access_log', 'a+')
f2.write('Attack detected')
```

5.1.4. Đóng file đã mở

Sau khi hoàn tất các thao tác đọc ghi file thì gọi phương thức `close()` để đóng file đã mở. Ví dụ:

```
f1.close()
f2.close()
```

5.1.5. Đổi tên file

Sử dụng phương thức `os.rename(old, new)` để đổi tên một file. Ví dụ:

```
import os
os.rename('test.txt', 'test_new.txt')
```

5.1.6. Xóa file

Sử dụng phương thức `os.remove(file)` để xóa một file

khỏi hệ thống. Ví dụ:

```
import os
os.remove('test.txt')
```

5.2. Thư mục (Directory)

5.2.1. Tạo thư mục

Sử dụng phương thức `os.mkdir(dir)` để tạo thư mục. Ví dụ:

```
import os
os.mkdir('test')
```

5.2.2. Xóa thư mục

Sử dụng phương thức `os.rmdir(dir)` để xóa một thư mục. Ví dụ:

```
import os
os.rmdir('test')
```

5.2.3. Đọc nội dung thư mục

Sử dụng phương thức `os.listdir(dir)` để lấy danh sách tập tin, thư mục của thư mục `dir`. Khi gọi sẽ trả về một mảng danh sách các tập tin, thư mục. Ví dụ:

```
import os
allfiles = os.listdir('/root/downloads')
print allfiles
```

5.3. Module `os`

Module `os` là một module có nhiều phương thức hữu ích trong việc làm việc với các file và directory, như:

- `os.chdir(path)`: đổi thư mục hiện hành
- `os.getcwd()`: trả về thư mục hiện hành
- `os.chmod(path, mode)`: CHMOD một đường dẫn
- `os.chown(path, uid, gid)`: CHOWN một đường dẫn
- `os.makedirs(path[, mode])`: tạo đường dẫn (có recursive)

- `os.removedirs(path)`: xóa một đường dẫn (có recursive)

Xem thêm tại https://docs.python.org/2/library/os.html

5.4. Module `os.path`

Module `os.path` hỗ trợ các phương thức giúp thao tác nhanh chóng và thuận tiện hơn trên đường dẫn.

- `os.path.exists(path)`: kiểm tra 1 đường dẫn có tồn tại hay không
- `os.path.getsize(path)`: lấy file size (byte)
- `os.path.isfile(path)`: kiểm tra xem có phải là một file thông thường
- `os.path.isdir(path)`: kiểm tra xem có phải là một thư mục
- `os.path.dirname(path)`: trả về tên thư mục của path
- `os.path.getatime(path)`: trả về thời gian truy cập mới nhất

- `os.path.getmtime(path)`: trả về thời gian chỉnh sửa cuối cùng
- `os.path.getctime(path)`: trả về thời gian chỉnh sửa cuối của metadata trên một số hệ thống. Hoặc trả về thời gian tạo file trên Windows.

Xem thêm tại
https://docs.python.org/2/library/os.path.html

Chương 6. Xử lý hình ảnh

Sử dụng thư viện Pillow (PIL Fork) để tiến hành các thao tác xử lý hình ảnh đơn giản như resize, crop, rotate...

6.1. Cài đặt PIL

Vào trang web sau để download / cài đặt package PIL cho python của bạn
http://pillow.readthedocs.org/en/latest/installation.html

Sau khi cài đặt thì có thể sử dụng các module trong package IPL

```
from PIL import Image
```

6.2. Mở file

```
from PIL import Image
im = Image.open("photo.jpg")
```

Sau khi mở file hình thành công thì có thể thao tác trên đối tượng `im`.

6.3. Ghi file

Từ đối tượng `Image` có thể lưu file xuống máy tính bằng phương thức `save(path, type)`. Ví dụ:

```
...
im.save('photo_new.jpg', 'JPEG')
```

6.4. Tạo thumbnail

Sử dụng phương thức `thumbnail` như sau:

```
from PIL import Image
im = Image.open('photo.jpg')
im.thumbnail((100, 100))
im.save('photo_thumbnail.jpg', 'JPEG')
```

`thumbnail` không trả về image mới mà thực hiện trên object image đang gọi.

6.5. Các thao tác xử lý hình ảnh

Tham khảo thêm thư viện PIL tại:
http://pillow.readthedocs.org/en/latest/index.html

Chương 7. Xử lý file JSON

JSON là một trong những định dạng file trao đổi dữ liệu thông dụng nhất hiện nay. Với kiến trúc đơn giản và tương đồng với cấu trúc của Python nên việc thao tác JSON trên Python rất dễ hiểu.

7.1. Load file từ Internet

Thông thường dữ liệu JSON được lấy từ nguồn khác (như file, internet..) nên chương này sẽ bắt đầu bằng cách hướng dẫn download một file JSON từ Internet và sau đó mới parsing nội dung JSON download.

Sử dụng module `urllib2` để download file và module `json` để encode/decode JSON data. Ví dụ:

```
import urllib2
import json

response = urllib2.urlopen('https://api.github.com/
users/voduytuan/repos')

data = json.load(response)

print data
```

Ví dụ trên sẽ truy vấn đường dẫn https://api.github.com/users/voduytuan/repos để lấy danh sách Repository trên Github của mình dưới định dạng JSON. Sau đó, sẽ được

7.2. Parsing JSON Data

Nếu như bạn đã có JSON data dưới dạng chuỗi, muốn parsing chuỗi này thành Data thì sử dụng như cách dưới đây:

```
import json

mystring = '{"a":1,"b":2,"c":3,"d":4,"e":5}'
data = json.loads(mystring)
print data
(Hiển thị: {u'a': 1, u'c': 3, u'b': 2, u'e': 5, u'd
': 4})
```

7.3. Encoding JSON Data

Nếu như bạn đã có một biến và muốn encode thành JSON string thì có thể dùng theo cách sau:

```
import json

mydata = {
    'name': 'John',
    'age': 10
}
jsonstring = json.dumps(mydata)
print jsonstring
(hiển thị: {"age": 10, "name": "John"})
```

Chương 8. Xử lý file XML

Trong phần này, chúng ta sẽ parsing nội dung XML thành dữ liệu để xử lý. Để xử lý XML, ta sẽ sử dụng thư viện Beautifulsoup 4. Đây là một thư viện giúp việc triển khai việc parsing html, xml được nhanh chóng và tiện lợi.

8.1. Cài đặt Beautifulsoup

Bạn có thể tham khảo hướng dẫn cách cài đặt tại website http://www.crummy.com/software/BeautifulSoup/bs4/doc beautiful-soup.

Trên MacOS, có thể cài bằng `pip` như sau:

```
$ sudo pip install beautifulsoup4
```

8.2. Cài đặt `lxml` parser

Để parsing `xml` từ beautifulsoup, tao sử dụng bộ parser xml có tên là `lxml`. Xem hướng dẫn cài đặt tại

http://www.crummy.com/software/BeautifulSoup/bs4/doc/#a-parser

Trên MacOS, có thể cài bằng `pip` như sau:

```
sudo pip install lxml
```

8.3. Ví dụ về parsing XML

Cho ví dụ sau:

```
from bs4 import BeautifulSoup as Soup

note = '''
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>
    <food>
        <name>Belgian Waffles</name>
        <price>$5.95</price>
        <description>Two of our famous Belgian Waff
les with plenty of real maple syrup</description>
        <calories>650</calories>
    </food>
    <food>
        <name>Strawberry Belgian Waffles</name>
```

```
        <price>$7.95</price>
        <description>Light Belgian waffles covered
with strawberries and whipped cream</description>
        <calories>900</calories>
    </food>
</breakfast_menu>
'''

soup = Soup(note, 'xml')

foods = soup.findAll('food')

for x in foods:
    print x.find('name').string, ': ', x.price.stri
ng
```

Khi chạy thì sẽ hiển thị ra màn hình như sau:

```
Belgian Waffles :  $5.95
Strawberry Belgian Waffles :  $7.95
```

Đối tượng thuộc class `Soup` (BeautifulSoup) sẽ giúp truy xuất các thành phần của file xml nhanh chóng và tiện lợi.

Trong ví dụ có một số cách truy xuất đến các phần tử như:

- `findAll()`: trả về mảng các thẻ có tên cần tìm
- `find()`: trả về phần tử đầu tiên có tên cần tìm
- Truy xuất trực tiếp thông qua tên thẻ như `x.price.string`

8.4. Parsing HTML

Tương tự như `xml`, BeautifulSoup có thể parsing nội dung HTML thông qua hàm khởi tạo và chọn `html` ở tham số thứ 2.

```
...
soup = Soup(websitehtml, 'html')
```

Chương 9. Kết nối MySQL

MySQL là một hệ cơ sở dữ liệu quan hệ phổ biến nhất hiện nay. Rất nhiều ngôn ngữ có thể kết nối đến MySQL và Python cũng không ngoại lệ.

Mặc định Python không có thư viện để kết nối đến MySQL server. Trong phần này, để kết nối đến MySQL Server từ Python, chúng ta sẽ sử dụng module `MySQLdb`

9.1. Cài đặt MySQLdb

Cài đặt thông qua `pip` như sau:

```
$ sudo pip install MySQL-python
```

Tham khảo thêm tại: https://pypi.python.org/pypi/MySQL-python/1.2.5

Khai báo module MySQLdb là có thể sử dụng.

```
import MySQLdb
```

Trong một số trường hợp đã cài MySQLdb nhưng import báo lỗi thiếu file `libmysqlclient.18.dylib`. Nguyên nhân có thể là do hiện tại đường dẫn đến file không tồn tại. Thử tạo symlink hoặc sử dụng câu lệnh sau để tạo symlink từ file `libmysqlclient.18.dylib` đến thư mục `/usr/lib/`

```
$ sudo ln -s /usr/local/mysql/lib/libmysqlclient.18.dylib /usr/lib/libmysqlclient.18.dylib
```

9.2. Kết nối đến MySQL Server

Bạn cần có một MySQL server đang chạy và cho kết nối đến. Trong trường hợp này là localhost nên không cần cấu hình đặc biệt, chỉ cần cung cấp username và password là có thể kết nối đến MySQL Server.

Thực hiện gọi hàm như sau để trả về kết nối:

```
import MySQLdb

dbcon = MySQLdb.connect(host = 'localhost', user = 'myusername', passwd = 'mypassword', db = 'mydbname')
```

Nếu kết nối không thành công thì sẽ báo lỗi và ngưng chương trình. Cách kết nối an toàn là đưa vào trong `try` để bắt lỗi như sau:

```
import MySQLdb

db = None

try:
    db = MySQLdb.connect(host = 'localhost', user =
 'root', passwd = 'root', db = 'mysql')

except MySQLdb.Error, e:
    print "Error %d: %s" % (e.args[0],e.args[1])
    sys.exit(1)

if db:
    cur = db.cursor()
    cur.execute("SELECT VERSION()")
    ver = cur.fetchone()
    print "Database version : %s " % ver
```

9.3. Kết nối với `charset utf8`

Mặc định kết nối đến MySQL server là charset `latin` nên khi lấy dữ liệu unicode thì hiển thị Tiếng Việt không đúng, bạn cần phải chọn charset là `utf8` khi tạo kết nối. Cú pháp như sau:

```
...
db = MySQLdb.connect(host = 'localhost', user = 'root', passwd = 'root', db = 'test', charset = 'utf8')
```

9.4. Query dữ liệu

Để truy vấn dữ liệu (chạy câu lệnh SQL) thì sử dụng `cursor` của MySQLdb. Ví dụ:

```
import MySQLdb

db = MySQLdb.connect(host = 'localhost', user = 'root', passwd = 'root', db = 'mysql');
cursor = db.cursor()
sql = 'SELECT * FROM user'
cursor.execute(sql)
myusers = cursor.fetchall()
```

Ví dụ dữ liệu biến `myusers` là: `((1, 'John'), (2, 'Doe'))`

Mặc định, `cursor` sẽ trả về mỗi dòng dữ liệu từ MySQL theo kiểu `tuple`, tức là không có key. Nếu bạn muốn trả về

kiểu `Dictionary` thì có thể khai báo:

```
import MySQLdb

db = MySQLdb.connect(host = 'localhost', user = 'root', passwd = 'root', db = 'mysql')
cursor = db.cursor(MySQLdb.cursors.DictCursor)
...
```

9.5. Fetch dữ liệu

Có một số cách để fetch dữ liệu thông dụng từ `cursor` sau khi đã `exectute(sql)` là `fetchone()` và `fetchall()`.

- `fetchone()`: chỉ fetch một dòng dữ liệu. Nếu muốn fetch nhiều dòng dữ liệu thì có thể gọi nhiều lần, mỗi lần sẽ trả về một dòng dữ liệu. Tốt cho trường hợp truy vấn rất nhiều dữ liệu một lúc. Nếu trả về `None` tức là đã fetch hết dữ liệu từ câu truy vấn.
- `fetchall()`: fetch toàn bộ dữ liệu truy vấn được từ câu truy vấn và trả về một tuple chứa các dòng dữ

liệu. Mỗi phần tử của Tuple có thể là một Tuple khác hoặc là một Dictionary tùy theo cài đặt ở bước lấy cursor (Xem phần 9.4).

- `fetchmany(size)`: nằm ở giữa 2 kiểu fetch trên, có thể quy định số lượng row trong mỗi lần fetch. Nếu fetch hết thì trả về Tuple rỗng.

9.6. Đóng kết nối

Sau khi kết nối và truy vấn thì có thể đóng kết nối theo ví dụ sau:

```
import MySQLdb

db = MySQLdb.connect(...)
db.close()
```

Ngoài ra, bạn cũng nên đóng `cursor` khi không còn sử dụng theo ví dụ:

```
import MySQLdb

db = MySQLdb.connect(...)
cursor = db.cursor()
cursor.close()
db.close()
```

9.7. Prepared Statement

Prepared statement là kỹ thuật tham số hóa các dữ liệu truyền vào câu truy vấn thay vì nối chuỗi trực tiếp để xây dựng một chuỗi truy vấn dài. Kỹ thuật này được áp dụng nhiều và giúp tăng hiệu quả và tính bảo mật của câu truy vấn. Ví dụ:

```
...
cur.execute("UPDATE Writers SET Name = %s WHERE Id
= %s", ("John", "4"))
...
```

Mỗi tham số truy vấn sẽ được thay thế bằng `%s` và phương thức `execute()` sẽ có tham số thứ 2 là một Tuple có giá trị

tương ứng với thứ tự xuất hiện của các thành phần `%s`

Chương 10. Kết nối Redis

Redis là một memory cache server hỗ trợ persistant data thông dụng nhất hiện nay. Nội dung chương này sẽ hướng dẫn bạn kết nối đến một Redis server (đã được cài đặt sẵn) thông qua thư viện redis-py.

10.1. Cài đặt

Có thể xem thêm về hướng dẫn cài đặt thư viện này tại https://github.com/andymccurdy/redis-py

Đơn giản cài thông qua `pip` là:

```
$ sudo pip install redis
```

10.2 Kết nối đến Redis

Để kết nối đến Redis server thì bạn có thể xem ví dụ sau:

```
import redis

r = redis.StrictRedis(host='localhost', port=6379,
db=0)
```

10.3. Thực hiện lệnh

Thực hiện các lệnh bình thường trên đối tượng redis. Ví dụ:

```
import redis

r = redis.StrictRedis(...)
r.set('foo', 'bar')
print r.get('foo')
(Hiển thị 'bar')
```

10.4. Pipeline

Pipeline là kỹ thuật được dùng trong trường hợp bạn muốn tăng performance bởi gộp nhiều lệnh vào một request thay vì mỗi lệnh là một request như thông thường. Xem ví dụ sau để hiểu cách sử dụng pipeline bằng `redis-py`:

```
import redis

r = redis.StrictRedis(...)
r.set('foo', 'bar')
pipe = r.pipeline()
pipe.set('a', 1)
pipe.set('b', 2)
pipe.set('c', 3)
pipe.get('foo')
pipe.execute()
```

Sau khi gọi phương thức `execute()` thì sẽ trả về List tương ứng với các kết quả của từng lệnh. Ví dụ kết quả từ đoạn code trên:

```
[True, True, True, 'bar']
```

Chương 11. Kết nối Memcached

Memcached là một memory cache server thông dụng hiện nay. Nội dung chương này sẽ hướng dẫn bạn kết nối đến một Memcached server (đã được cài đặt sẵn) thông qua thư viện `pylibmc`.

11.1. Cài đặt

Có thể xem thêm về hướng dẫn cài đặt thư viện này tại http://sendapatch.se/projects/pylibmc/install.html

Đơn giản cài thông qua `pip` là:

```
$ sudo pip install pylibmc
```

11.2 Kết nối đến Memcached Server

Để kết nối đến Memcached server thì bạn có thể xem ví dụ sau:

```
import pylibmc

mc = pylibmc.Client(["127.0.0.1"], binary=True, beh
aviors={"tcp_nodelay": True, "ketama": True})
```

10.3. Thực hiện lệnh

Thực hiện các lệnh bình thường trên đối tượng memcache. Ví dụ:

```
import pylibmc

mc = pylibmc.Client(...)
mc.set('foo', 'bar')
print mc.get('foo')
(Hiển thị 'bar')
```

Chương 12. Kết nối RabbitMQ

RabbitMQ là một phần mềm cho phép xây dựng Message Queue theo protocol AMQP và khá thông dụng trên thế giới. Để kết nối đến RabbitMQ trên Python, ta sẽ sử dụng thư viện `pika`.

12.1. Cài đặt

Có thể xem thêm về hướng dẫn cài đặt thư viện này tại https://pika.readthedocs.org/en/0.10.0/

Đơn giản cài thông qua `pip` là:

```
$ sudo pip install pika
```

12.2 Gởi một message đến Server - Provider

Ví dụ để gởi một message đến server

```
import pika

connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')

channel.basic_publish(exchange='', routing_key='hello', body='Hello World!')
print " [x] Sent 'Hello World!'"

connection.close()
```

Để gởi một message, chúng ta cần kết nối đến server và khai báo một channel, ở đây là channel có tên là `hello`. Sau khi khai báo channel, tao tiến hành gởi message có nội dung `Hello World!` thông qua channel này, kèm theo khai báo `routing_key` là `hello`. Routing Key sẽ giúp điều hướng message này đến đúng các worker được khai báo nhận message theo routing key (Consumer)

12.3. Nhận message - Consumer

Ở bước trước, ta đã tạo một message lên queue. Ở bước này, ta sẽ khai báo một worker xử lý các message nhận được từ channel `hello`.

```
import pika

connection = pika.BlockingConnection(pika.Connectio
nParameters(host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='hello')

print ' [*] Waiting for messages. To exit press CTR
L+C'

def callback(ch, method, properties, body):
    print " [x] Received %r" % (body,)

channel.basic_consume(callback, queue='hello', no_a
ck=True)

channel.start_consuming()
```

Đoạn code này cũng có phần khai báo `connection`,

`channel`. Tuy nhiên, vì là consumer nên sẽ sử dụng phương thức `basic_consume` để lắng nghe trên queue `hello`, khi có message đến sẽ gọi hàm `callback()` để xử lý. Trong trường hợp ví dụ này thì callback chỉ đơn giản là hiển thị chuỗi thông báo đã nhận được message.

12.4. Tìm hiểu thêm về RabbitMQ và Pika

Nội dung chương này không có ý định giới thiệu tất cả khái niệm của AMQP cũng như RabbitMQ mà chủ yếu giới thiệu thư viện `pika` để làm việc với RabbitMQ. Bạn có thể tiếp tục theo dõi các hướng dẫn về khái niệm của RabbitMQ và Pika tại website chính thức của RabbitMQ.

https://www.rabbitmq.com/tutorials/tutorial-one-python.html

Chương 13. Restful client

Ngày nay, với sự phong phú của các ứng dụng và nhu cầu kết nối ngày càng lớn thì Restful là một trong những mô hình web service được nhiều công ty sử dụng. Do đó, việc có thể kết nối đến các Restful web service là một kỹ năng cần thiết cho các lập trình viên.

Chương này sẽ giới thiệu về một trong những thư viện nổi tiếng nhất trong Python để làm việc với các Restful web service, đó là `requests`.

Website chính thức và các mô tả đầy đủ đều được đề cập tại http://www.python-requests.org/

13.1. Cài đặt

Cài đặt nhanh chóng thông qua `pip` như sau:

```
$ sudo pip install requests
```

13.2. Request

Hỗ trợ sẵn các phương thức tương ứng cho Http request như GET, POST, PUT, DELETE...

```
import requests

r = requests.get('https://api.github.com/events')
r = requests.post("http://httpbin.org/post")
r = requests.put("http://httpbin.org/put")
r = requests.delete("http://httpbin.org/delete")
r = requests.head("http://httpbin.org/get")
r = requests.options("http://httpbin.org/get")
```

13.2.1. GET Query string

Đối với các request `GET`, có thể truyền tham số đường dẫn thông qua tham số `params` khi gọi phương thức `get()`. Ví dụ:

```
import requests

payload = {'key1': 'value1', 'key2': 'value2'}
r = requests.get("http://httpbin.org/get", params =
 payload)
print(r.url)
(Hiển thị: http://httpbin.org/get?key2=value2&key1=
value1)
```

13.2.2. Request Body

Trong hầu hết trường hợp các request như POST, PUT thương cần truyền dữ liệu khi request, có thể sử dụng tham số `data` để truyền data lên kèm request. Ví dụ:

```
import requests

payload = {'key1': 'value1', 'key2': 'value2'}
r = requests.post("http://httpbin.org/post", data =
 payload)
```

13.2.3. Upload file

Có thể gởi thêm tham số `files` để upload file kèm theo request. Ví dụ:

```
import requests

url = 'http://httpbin.org/post'
files = {'file': open('report.xls', 'rb')}
r = requests.post(url, files=files)
```

13.3. Response

Sau khi gọi các phương thức tương ứng để request, bạn sẽ có đối tượng thuộc class `Response`. Đối tượng này có một số thông tin như sau:

- `status_code`: HTTP Status server trả về
- `headers`: Các thông tin header mà server trả về dưới dạng Dictionary.
- `cookies`: Nếu server có trả về cookie thì có thể sử dụng thuộc tính này để lấy các cookie.
- `text`: Trả về nội dung response

Tham khảo về các tính năng khác của thư viện `requests` tại website chính thức. http://docs.python-requests.org

Chương 14. Gởi email với SMTP

SMTP là giao thức gởi mail thông dụng hiện nay. Python hỗ trợ mặc định thư viện `smtplib` dùng để kết nối đến một SMTP Server và gởi email. Tuy nhiên, việc sử dụng thư viện này sẽ gây khó khăn cho việc định dạng và sử dụng nên chúng ta sẽ sử dụng thư viện `sender`, là một thư viện giúp định dạng và gởi email đơn giản hơn.

14.1. Cài đặt `sender`

Cài đặt từ `pip` như sau:

```
$ sudo pip install sender
```

14.2. Gởi email đơn giản

Để gởi 1 email với `sender`, bạn cần có tài khoản và một số thông tin của SMTP Server trước khi gởi. Ví dụ đoạn code để gởi 1 email từ SMTP Server của Amazon.

```
from sender import Mail, Message

mail = Mail(
    "smtp.gmail.com",
    port = 465,
    username = "example@gmail.com",
    password = "yourpassword",
    use_tls = False,
    use_ssl = True,
    debug_level = False
)

msg = Message("msg subject")
msg.fromaddr = ("Vo Duy Tuan", "example@gmail.com")
msg.to = "destuser@gmail.com"
msg.body = "this is a msg plain text body"
msg.html = "<b>this is a msg text body</b>"
msg.reply_to = "example@gmail.com"
msg.charset = "utf-8"
msg.extra_headers = {}
msg.mail_options = []
msg.rcpt_options = []

# Send message
mail.send(msg)
```

14.3. Gởi email có đính kèm file

Bạn cần sử dụng thêm class Attachment để tạo attachment.

```
from sender import Mail, Message, Attachment

mail = Main(...)
msg = Message(..)
...

# Open attached file and create Attachment object
with open("photo01.jpg") as f:
    file01 = Attachment("photo01.jpg", "image/jpeg"
, f.read())

msg.attach(file01)

# Send message
mail.send(msg)
```

14.4. Tìm hiểu thêm

Bạn có thể tìm hiểu thêm về thư viện `sender` tại website

chính thức tại địa chỉ http://sender.readthedocs.org/

Chương 15. Socket programming

Chương này sẽ ví dụ việc xây dựng một môi trường Client - Server sử dụng Socket. Server sẽ lắng nghe trên một port (12345) và khi client kết nối vào sẽ thông báo hiển thị thông tin của client (IP và Port) và gởi 1 message xuống cho client.

15.1. Server side

Tạo file `server.py` với nội dung bên dưới.

```
import socket

s = socket.socket()
host = socket.gethostname()
port = 12345
s.bind((host, port))

s.listen(5)
while True:
   c, addr = s.accept()
   print 'Got connection from', addr
   c.send('Thank you for connecting')
   c.close()
```

Đoạn code trên khi thực thi sẽ chạy và lắng nghe ở port TCP 12345. Mỗi khi có một kết nối từ client sẽ hiện ra thông báo kết nối từ IP và Port nào, ví dụ: `Got connection from Got connection from ('192.168.1.104', 60018)`. Sau đó, gởi trả một message với nội dung `Thank you for connecting` về cho client. Sau đó, đóng kết nối với client.

15.2. Client side

Tạo file `client.py` với nội dung bên dưới.

```
import socket

s = socket.socket()
host = '127.0.0.1'
port = 12345

s.connect((host, port))
print s.recv(1024)
s.close
```

Đoạn code trên sẽ kết nối đến một socket server thông qua hostname lấy được từ phương thức `socket.gethostname()` và port 12345. Sau khi kết nối, sẽ hiển thị ra kết quả trả về từ server. Sau đó thì đóng kết nối.