Ghi chú của một coder

Vũ Anh

Tháng 01 năm 2018

Mục lục

Phần I

Lập trình

Chương 1

Python

Hướng dẫn online tại http://magizbox.com/training/python/site/

01/11/2017
Thích python vì nó quá đơn giản (và quá đẹp).

1.1 Khóa học 1: Nhập môn Python

1.1.1 Mục tiêu của khóa học

Ưu điểm của khóa học

- Dành cho người mới bắt đầu, chưa từng học lập trình hoặc cho những ai muốn ôn lại kiến thức căn bản về lập trình python.
- Dễ học, dễ thực hành, ví dụ trực quan thú vị, không yêu cầu cao về máy móc hay phần mềm đi kèm.
- Ví dụ mẫu nhiều, trực quan, thú vị.

Kết thúc khóa học bạn sẽ học được gì?

- Xây dựng 5 dự án đơn giản với Python 3

Với những kiến thức bạn có thể làm gì?

- Lập trình viên Python tại các công ty phần mềm

1.1.2 Đối tượng học viên

- Những bạn chưa từng lập trình
- Những bạn đã có kinh nghiệm lập trình nhưng chưa lập trình python

1.2 Giới thiệu

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985-1990. Like Perl, Python source code is also available under the GNU General

Public License (GPL). This tutorial gives enough understanding on Python programming language.
Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
Python is Interactive: You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
Python is Object-Oriented: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
Python is Beginner Friendly: Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.
Sách
Tập hợp các sách python
Khoá học
Tập hợp các khóa học python
Tham khảo
Top 10 Python Libraries Of 2015

1.3 Cài đặt

1.3.1 Trên Windows

Anaconda 4.3.0
Anaconda is BSD licensed which gives you permission to use Anaconda commercially and for redistribution.
1. Download the installer
2. Optional: Verify data integrity with MD5 or SHA-256 3. Double-click the .exe file to install Anaconda and follow the instructions on the screen
Python 3.6 version 64-BIT INSTALLER Python 2.7 version 64-BIT INSTALLER
Step 2. Discover the Map
https://docs.python.org/2/library/index.html

1.3.2 Trên CentOS

Developer tools
The Development tools will allow you to build and compile software from source code. Tools for building RPMs are also included, as well as source code management tools like Git, SVN, and CVS.

```
yum groupinstall "Development tools"
yum install zlib-devel
yum install bzip2-devel
yum install openssl-devel
yum install ncurses-devel
yum install sqlite-devel
```

Python Anaconda Anaconda is BSD licensed which gives you permission to use Anaconda commercially and for redistribution.

```
cd /opt
wget --no-check-certificate https://www.python.org/ftp/python/2.7.6/
    ↪ Python-2.7.6.tar.xz
tar xf Python-2.7.6.tar.xz
cd Python-2.7.6
./configure --prefix=/usr/local
make && make altinstall
## link
ln -s /usr/local/bin/python2.7 /usr/local/bin/python
# final check
which python
python -V
# install Anaconda
cd ~/Downloads
wget https://repo.continuum.io/archive/Anaconda-2.3.0-Linux-x86_64
    ↪ .sh
bash ~/Downloads/Anaconda-2.3.0-Linux-x86_64.sh
```

1.4 Biến - Hộp nhỏ

1.5 123s - Số trong Python

1.5.1 Print, print

```
print "Hello World"
```

1.5.2 Conditional

```
if you_smart:
    print "learn python"
else:
    print "go away"
```

1.5.3 Loop

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.
Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement
Python programming language provides following types of loops to handle looping requirements.
while loop Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body. for loop Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. nested loops You can use one or more loop inside any another while, for or do..while loop.

1.5.4 While Loop

A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.
Syntax
The syntax of a while loop in Python programming language is

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

Example

```
count = 0
while count < 9:
   print 'The count is:', count
   count += 1
print "Good bye!"
```

1.5.5 For Loop

It has the ability to iterate over the items of any sequence, such as a list or a string.
Syntax

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

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable iterating$_var.Next, thestatementsblockisexecuted.Eachiter$
Example

```
for i in range(10):
    print "hello", i

for letter in 'Python':
   print 'Current letter :', letter

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

print "Good bye!"
```

Yield and Generator
Yield is a keyword that is used like return, except the function will return a generator.

```
def createGenerator():
    yield 1
    yield 2
    yield 3
mygenerator = createGenerator() # create a generator
print(mygenerator) # mygenerator is an object!
# <generator object createGenerator at 0xb7555c34>
```

```
for i in mygenerator:
    print(i)
# 1
# 2
# 3
```

Visit Yield and Generator explained for more information
Functions
Variable-length arguments

```
def functionname([formal_args,] *var_args_tuple ):
   "function_docstring"
   function_suite
   return [expression]
```

Example

```
#!/usr/bin/python

# Function definition is here
def printinfo( arg1, *vartuple ):
   "This prints a variable passed arguments"
   print "Output is: "
   print arg1
   for var in vartuple:
      print var
   return;

# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

Coding Convention Code layout Indentation: 4 spaces
Suggest Readings
"Python Functions". www.tutorialspoint.com "Python Loops". www.tutorialspoint.com "What does the “yield” keyword do?". stackoverflow.com "Improve Your Python: ’yield’ and Generators Explained". jeffknupp.com
Vấn đề với mảng

Random Sampling [1] - sinh ra một mảng ngẫu nhiên trong khoảng (0, 1), mảng ngẫu nhiên số nguyên trong khoảng (x, y), mảng ngẫu nhiên là permutation của số từ 1 đến n

1.6 Cấu trúc dữ liệu

1.6.1 Number

Basic Operation

[1] tham khảo [pytorch](http://pytorch.org/docs/master/torch.html?highlight=randntorch.randn), [numpy](https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.random.html))

1
1.2
1 + 2
abs(−5)

1.6.2 Collection

In this post I will cover 4 most popular data types in python list, tuple, set, dictionary
List The most basic data structure in Python is the sequence. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.
The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.
Usage
A list keeps order, dict and set don't: when you care about order, therefore, you must use list (if your choice of containers is limited to these three, of course) Most Popular Operations
Create a list a = ["a", "b", 3] Access values in list a[1] Updated List a[0] = 5 Delete list elements del a[1] Reverse a list a[::-1] Itertools [a + b for (a, b) in itertools.product(x, y)] Select random elements in list random.choice(x) random.sample(x, 3) Create a list a = [1, 2, 3] [1, 2, 3] Access values in list list1 = ['physics', 'chemistry', 1997, 2000] list2 = [1, 2, 3, 4, 5, 6, 7]
print list1[0] physics
print list2[1:5] [2, 3, 4, 5] Updated lists list = ['physics', 'chemistry', 1997, 2000]
print list[2] 1997
list[2] = 2001 print list[2] 2001 Delete list elements list1 = ['physics', 'chemistry', 1997, 2000];
print list1 ['physics', 'chemistry', 1997, 2000]
del list1[2]
print list1 ['physics', 'chemistry', 2000] Reverse a list [1, 3, 2][::-1] [2, 3, 1] Itertools import itertools
x = [1, 2, 3] y = [2, 4, 5]
[a + b for (a, b) in itertools.product(x, y)] [3, 5, 6, 4, 6, 7, 5, 7, 8] Select random elements in list import random
x = [13, 23, 14, 52, 6, 23]
random.choice(x) 52
random.sample(x, 3) [23, 14, 52] Tuples A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.
Usage
Tuples have structure, lists have order Tuples being immutable there is also a semantic distinction that should guide their usage. Tuples are heterogeneous data structures (i.e., their entries have different meanings), while lists are homogeneous sequences Most Popular Operations
Create a tuple t = ("a", 1, 2) Accessing Values in Tuples t[0], t[1:] Updating Tuples Not allowed Create a tuple tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5); tup3 = "a", "b", "c", "d"; tup4 = () tup5 = (50,) Accessing Values in Tuples !/usr/bin/python
tup1 = ('physics', 'chemistry', 1997, 2000); tup2 = (1, 2, 3, 4, 5, 6, 7);
tup1[0] physics
tup2[1:5] [2, 3, 4, 5] Updating Tuples Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates
tup1 = (12, 34.56); tup2 = ('abc', 'xyz');
Following action is not valid for tuples tup1[0] = 100;
So let's create a new tuple as follows tup3 = tup1 + tup2; print tup3 Set Sets are lists with no duplicate entries.
The sets module provides classes for constructing and manipulating unordered collections of unique elements. Common uses include membership testing, removing duplicates from a sequence, and computing standard math operations on sets such as intersection, union, difference, and symmetric difference.
Usage
set forbids duplicates, list does not: also a crucial distinction. Most Popular Operations
Create a set x = set(["Postcard", "Radio", "Telegram"]) Add elements to a set x.add("Mobile") Remove elements to a set x.remove("Radio") Subset y.issubset(x) Intersection x.intersection(y) Difference between two sets x.difference(y)
Create a set x = set(["Postcard", "Radio", "Telegram"]) x set(['Postcard', 'Telegram', 'Radio']) Add elements to a set x = set(["Postcard", "Radio", "Telegram"]) x.add("Mobile") x set(['Postcard', 'Telegram', 'Mobile', 'Radio']) Remove elements to a set x = set(["Postcard", "Radio", "Telegram"]) x.remove("Radio") x set(['Postcard', 'Telegram']) Subset x = set(["a","b","c","d"]) y = set(["c","d"]) y.issubset(x) True Intersection x = set(["a","b","c","d"]) y = set(["c","d"]) x.intersection(y) set(['c', 'd']) Difference between two sets x = set(["Postcard", "Radio", "Telegram"]) y = set(["Radio","Television"]) x.difference(y) set(['Postcard', 'Telegram']) Dictionary Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: .
Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.
Usage
dict associates with each key a value, while list and set just contain values: very different use cases, obviously. Most Popular Operations
Create a dictionary d = "a": 1, "b": 2, "c": 3 Update dictionary d["a"] = 4 Delete dictionary elements del d["a"] Create a dictionary dict = 'Name': 'Zara', 'Age': 7, 'Class': 'First'
print "dict['Name']: ", dict['Name'] print "dict['Age']: ", dict['Age'] Update dictionary dict = 'Name': 'Zara', 'Age': 7, 'Class': 'First'
dict['Age'] = 8; update existing entry dict['School'] = "DPS School"; Add new entry
print "dict['Age']: ", dict['Age'] print "dict['School']: ", dict['School'] Delete dictionary elements dict = 'Name': 'Zara', 'Age': 7, 'Class': 'First'

del dict['Name']; remove entry with key 'Name' dict.clear(); remove all entries in dict del dict ; delete entire dictionary
print "dict['Age']: ", dict['Age'] print "dict['School']: ", dict['School'] Related Readings Python Lists, tutorialspoint.com Python Dictionary, tutorialspoint.com Python Dictionary Methods, guru99 In Python, when to use a Dictionary, List or Set?, stackoverflow What's the difference between lists and tuples?, stackoverflow

1.6.3 String

Format '0, 1, 2'.format('a', 'b', 'c') 'a, b, c' Regular Expressions The aim of this chapter of our Python tutorial is to present a detailed led and descriptive introduction into regular expressions. This introduction will explain the theoretical aspects of regular expressions and will show you how to use them in Python scripts.
Regular Expressions are used in programming languages to filter texts or textstrings.
It's possible to check, if a text or a string matches a regular expression.
There is an aspect of regular expressions which shouldn't go unmentioned: The syntax of regular expressions is the same for all programming and script languages, e.g. Python, Perl, Java, SED, AWK and even X.
Functions match function This function attempts to match RE pattern to string with optional flags.
re.match(pattern, string, flags=0) Example
import re
line = "Cats are smarter than dogs"
$matched_object = re.match(r'(.*)are(.*?).*', line, re.M|re.I)$
$if\ matched_object : print"matched_object.group() : ", matched_object.group()print"matched_object.group(1) :$
$", matched_object.group(1)print"matched_object.group(2) : ", matched_object.group(2)else :$
$print"Nomatch!!"Whenthecodeisexecuted, itproducesfollowingresults$
$matched_object.group() : Catsaresmarterthandogsmatched_object.group(1) : Catsmatched_object.group(2) :$
$smartersearchfunctionThisfunctionsearchesforfirstoccurrenceofREpatternwithinstirngwithoptionalf$
re.search(pattern, string, flags=0) Example
!/usr/bin/python import re
line = "Cats are smarter than dogs"
$search_object = re.search(r'dogs', line, re.M|re.I)if\ search_object : print"search-$
$- > search_object.group() : ", search_object.group()else : print"Nothingfound!!"Whenthecodeisexecuted, it$
search -> $search_object.group() : dogssubfunctionThismethodreplacesalloccurrencesoftheREpatterninstri$
re.sub(pattern, repl, string, max=0) Example
!/usr/bin/python import re
phone = "2004-959-559 This is Phone Number"
Delete Python-style comments num = re.sub(r'.*', "", $phone)print"PhoneNum :$
$", num$
Remove anything other than digits num = re.sub(r", "", phone) print "Phone Num : ", num When the code is executed, it produces following results
Phone Num : 2004-959-559 Phone Num : 2004959559 Tokens Cheatsheet Character Classes . any character except newline /go.gle/ google goggle gogle .word, digit, whitespace // AaYyz09 ?! // 012345 aZ? // 0123456789 abcd?/ §not word, digit, whitespace // abcded 1234 ?> // abc 12345 ?<. /§/ abc 123? <. [abc] any of a, b or c /analy[sz]e/ analyse analyze analyxe $[^abc]nota, borc/analy[^sz]e/analyseanalyzeanalyxe[a-g]characterbetweenag/[2-4]/demo1demo2demo3demo4demo5QuantifiersAlternationa*$

$a+a?0 or more, 1 or more, 0 or 1 /go*gle/goglegoglegooglegooooogleh gle/go+gle/gglegoglegooglegooooogleh gle/$
start / end of the string /^{a}bc/ abc /$^{a}bc/abcabc/abc$/ abc abc _word, not-word boundary /$\underline{\ }$/ This island is beautiful. // cat certificate Escaped characters $\dot{}$ escaped special characters /$\dot{/}$ username@exampe.com 300.000 USD // abc@/ / abc@ $\hat{t}$ab, linefeed, carriage return /$\widehat{/}$ abc def /ab/ ab /$\mathring{/}$ abc@ǒ0A9 unicode escaped © /ǒ0A9/ Copyright©2017 - All rights reserved Groups and Lockaround (abc) capture group /(demo|example)[0-9]/ demo1example4demo backreference to group 1 /(abc|def)=/ abc=abc def=defabc=def (?:abc) non-capturing group /(?:abc)3/ abcabcabc abcabc (?=abc) positive lookahead /t(?=s)/ tttsssttts (?!abc) negative lookahead /t(?!s)/ tttsssttts (?<=abc) positive lookbehind /(?<=foo)bar/ foobar fuubar (?<!abc) negative lookbehind /(?<!foo)bar/ foobar fuubar Related Readings
Online regex tester and debugger: PHP, PCRE, Python, Golang and JavaScript, regex101.com RegExr: Learn, Build, Test RegEx, regexr.com

1.6.4 Datetime

Print current time
from datetime import datetime datetime.now().strftime(' '2015-12-29 14:02:27' Get current time
import datetime datetime.datetime.now() datetime(2009, 1, 6, 15, 8, 24, 78915) Unixtime
import time int(time.time()) Measure time elapsed
import time
start = time.time() print("hello") end = time.time() print(end - start) Moment
Dealing with dates in Python shouldn't have to suck.
Installation
pip install moment Usage
import moment from datetime import datetime
Create a moment from a string moment.date("12-18-2012")
Create a moment with a specified strftime format moment.date("12-18-2012", "
Moment uses the awesome dateparser library behind the scenes moment.date("2012-12-18")
Create a moment with words in it moment.date("December 18, 2012")
Create a moment that would normally be pretty hard to do moment.date("2 weeks ago")
Create a future moment that would otherwise be really difficult moment.date("2 weeks from now")
Create a moment from the current datetime moment.now()
The moment can also be UTC-based moment.utcnow()
Create a moment with the UTC time zone moment.utc("2012-12-18")
Create a moment from a Unix timestamp moment.unix(1355875153626)
Create a moment from a Unix UTC timestamp moment.unix(1355875153626, utc=True)
Return a datetime instance moment.date(2012, 12, 18).date
We can do the same thing with the UTC method moment.utc(2012, 12, 18).date
Create and format a moment using Moment.js semantics moment.now().format("YYYY-M-D")

Create and format a moment with strftime semantics moment.date(2012, 12, 18).strftime("
Update your moment's time zone moment.date(datetime(2012, 12, 18)).locale("US/Central").date
Alter the moment's UTC time zone to a different time zone moment.utcnow().timezone("US/Eastern").date
Set and update your moment's time zone. For instance, I'm on the west coast, but want NYC's current time. moment.now().locale("US/Pacific").timezone("US/Eastern")
In order to manipulate time zones, a locale must always be set or you must be using UTC. moment.utcnow().timezone("US/Eastern").date
You can also clone a moment, so the original stays unaltered now = moment.utcnow().timezone("US/Pacific") future = now.clone().add(weeks=2) Related Readings How to get current time in Python, stackoverflow Does Python's time.time() return the local or UTC timestamp?, stackoverflow Measure time elapsed in Python?, stackoverflow momnet, https://github.com/zachwill/moment

1.6.5 Object

Convert dict to object Elegant way to convert a normal Python dict with some nested dicts to an object
class Struct: def $_{init_{(self,**entries):self._{dict_{.update(entries)Then,youcanuse}}}}$
> args = 'a': 1, 'b': 2 > s = Struct(**args) > s < $_{main_{.Structinstanceat0x01D6A738>>s.a1>s.b2RelatedReadings}}$
stackoverflow, Convert Python dict to object?

1.7 Lập trình hướng đối tượng

Object Oriented Programming Python has been an object-oriented language since it existed. Because of this, creating and using classes and objects are downright easy. This chapter helps you become an expert in using Python's object-oriented programming support.
If you do not have any previous experience with object-oriented (OO) programming, you may want to consult an introductory course on it or at least a tutorial of some sort so that you have a grasp of the basic concepts.

1.7.1 Classes and Objects

Classes can be thought of as blueprints for creating objects. When I define a BankAccount class using the class keyword, I haven't actually created a bank account. Instead, what I've created is a sort of instruction manual for constructing "bank account" objects. Let's look at the following example code:

```
class BankAccount:
    id = None
    balance = 0

    def __init__(self, id, balance=0):
        self.id = id
        self.balance = balance

    def __get_balance(self):
        return self.balance
```

```
    def withdraw(self, amount):
        self.balance = self.balance - amount

    def deposit(self, amount):
        self.balance = self.balance + amount

john = BankAccount(1, 1000.0)
john.withdraw(100.0)
```

The class BankAccount line does not create a new bank account. That is, just because we've defined a BankAcount doesn't mean we've created on; we've merely outlined the blueprint to create a BankAccount object. To do so, we call the class's $_{init_{methodwiththeproper number of arguments (minus self, which we'll get to in a moment)}}$
So, to use the "blueprint" that we crated by defining the class BankAccount (which is used to create BankAccount objects), we call the class name almost as if it were a function: john = BankAccount(1, 1000.0). This line simple say "use the BankAccount blueprint to create me a new object, which I'll refer to as john".
The john object, known as an instance, is the realized version of the BankAccount class. Before we called BankAccount(), no BankAccount object existed. We can, of course, create as many BankAccount objects as we'd like. There is still, however, only one BankAccount class, regardless of how many instances of the class we create.

1.7.2 self

So what's with that self parameter to all of the BankAccount methods? What is it? Why, it's the instance, of course! Put another way, a method like withdraw defines the instructions for withdrawing money from some abstract customer's account. Calling john.withdraw(100) puts those instructions to use on the john instance.
So when we say def withdraw(self, amount):, we're saying, "here's how you withdraw money from a BankAccount object (which we'll call self) and a dollar figure (which we'll call amount). self is the instance of the BankAccount that withdraw is being called on. That's not me making analogies, either. john.withdraw(100.0) is just shorthand for BankAccount.withdraw(john, 100.0), which is perfectly valid (if not often seen) code.
Constructors: $_{init}$
self may make sense for other methods, but what about $_{init_{?When we call }init_{, we're in the process of creating an object, so how can th}}$
This is why when we call $_{init_{, we initialize objects by saying things like self.id = id. Remember, since self is the instance, this is equivalent to saying i}}$
Be careful what you $_{init}$
After $_{init_{has finished, the caller can rightly assume that the object is ready to use. That is, after john = BankAccount(1,1000.0), we can start making depo}}$
Inheritance While Object-oriented Programming is useful as a modeling tool, it truly gains power when the concept of inheritance is introduced. Inheritance is the process by which a "child" class derives the data and behavior of a "parent" class. An example will definitely help us here.
Imagine we run a car dealership. We sell all types of vehicles, from motorcycles to trucks. We set ourselves apart from the competition by our prices. Specifically, how we determine the price of a vehicle on our lot: $5,000 x number of wheels

a vehicle has. We love buying back our vehicles as well. We offer a flat rate - 10% of the miles driven on the vehicle. For trucks, that rate is $10,000. For cars, $8,000. For motorcycles, $4,000.
If we wanted to create a sales system for our dealership using Object-oriented techniques, how would we do so? What would the objects be? We might have a Sale class, a Customer class, an Inventory class, and so forth, but we'd almost certainly have a Car, Truck, and Motorcycle class.
What would these classes look like? Using what we've learned, here's a possible implementation of the Car class:

```
class Car(object):
    def __init__(self, wheels, miles, make, model, year, sold_on):
        self.wheels = wheels
        self.miles = miles
        self.make = make
        self.model = model
        self.year = year
        self.sold_on = sold_on

    def sale_price(self):
        if self.sold_on is not None:
            return 0.0  # Already sold
        return 5000.0 * self.wheels

    def purchase_price(self):
        if self.sold_on is None:
            return 0.0  # Not yet sold
        return 8000 - (.10 * self.miles)
```

OK, that looks pretty reasonable. Of course, we would likely have a number of other methods on the class, but I've shown two of particular interest to us: $\text{sale}_price and purchase_price. We'll see why these are important in a bit.$
Now that we've got the Car class, perhaps we should create a Truck class? Let's follow the same pattern we did for car:

```
class Truck(object):
    def __init__(self, wheels, miles, make, model, year, sold_on):
        self.wheels = wheels
        self.miles = miles
        self.make = make
        self.model = model
        self.year = year
        self.sold_on = sold_on

    def sale_price(self):
        if self.sold_on is not None:
            return 0.0  # Already sold
        return 5000.0 * self.wheels

    def purchase_price(self):
        if self.sold_on is None:
```

```
            return 0.0  # Not yet sold
        return 10000 - (.10 * self.miles)
```

Wow. That's almost identical to the car class. One of the most important rules of programming (in general, not just when dealing with objects) is "DRY" or "Don't Repeat Yourself. We've definitely repeated ourselves here. In fact, the Car and Truck classes differ only by a single character (aside from comments). So what gives? Where did we go wrong? Our main problem is that we raced straight to the concrete: Car and Truck are real things, tangible objects that make intuitive sense as classes. However, they share so much data and functionality in common that it seems there must be an abstraction we can introduce here. Indeed there is: the notion of Vehicle.

1.7.3 Abstract Classes

A Vehicle is not a real-world object. Rather, it is a concept that some real-world objects (like cars, trucks, and motorcycles) embody. We would like to use the fact that each of these objects can be considered a vehicle to remove repeated code. We can do that by creating a Vehicle class:

```
class Vehicle(object):
    base_sale_price = 0

    def __init__(self, wheels, miles, make, model, year, sold_on):
        self.wheels = wheels
        self.miles = miles
        self.make = make
        self.model = model
        self.year = year
        self.sold_on = sold_on


    def sale_price(self):
        if self.sold_on is not None:
            return 0.0  # Already sold
        return 5000.0 * self.wheels

    def purchase_price(self):
        if self.sold_on is None:
            return 0.0  # Not yet sold
        return self.base_sale_price - (.10 * self.miles)
```

Now we can make the Car and Truck class inherit from the Vehicle class by replacing object in the line class Car(object). The class in parenthesis is the class that is inherited from (object essentially means "no inheritance". We'll discuss exactly why we write that in a bit).
We can now define Car and Truck in a very straightforward way:

```
class Car(Vehicle):

    def __init__(self, wheels, miles, make, model, year, sold_on):
        self.wheels = wheels
```

```
        self.miles = miles
        self.make = make
        self.model = model
        self.year = year
        self.sold_on = sold_on
        self.base_sale_price = 8000


class Truck(Vehicle):

    def __init__(self, wheels, miles, make, model, year, sold_on):
        self.wheels = wheels
        self.miles = miles
        self.make = make
        self.model = model
        self.year = year
        self.sold_on = sold_on
        self.base_sale_price = 10000
```

Object Convert dict to object

```
class Struct:
    def __init__(self, **entries):
        self.__dict__.update(entries)
```

Then, you can use

```
> args = {'a': 1, 'b': 2}
> s = Struct(**args)
> s
< __main__.Struct instance at 0x01D6A738 >
> s.a
1
> s.b
2
```

Suggested Readings Improve Your Python: Python Classes and Object Oriented Programming stackoverflow, Convert Python dict to object? Why are Python's 'private' methods not actually private?

1.7.4 Design Patterns

Design Patterns Singleton Non-thread-safe Paul Manta's implementation of singletons

```
@Singleton
class Foo:
    def __init__(self):
        print 'Foo created'

f = Foo() # Error, this isn't how you get the instance of a singleton
```

```
f = Foo.Instance() # Good. Being explicit is in line with the Python
    ↪ Zen
g = Foo.Instance() # Returns already created instance

print f is g # True

class Singleton:
    """
    A non-thread-safe helper class to ease implementing singletons.
    This should be used as a decorator -- not a metaclass -- to the
    class that should be a singleton.

    The decorated class can define one `__init__` function that
    takes only the `self` argument. Also, the decorated class cannot be
    inherited from. Other than that, there are no restrictions that
        ↪ apply
    to the decorated class.

    To get the singleton instance, use the `Instance` method. Trying
    to use `__call__` will result in a `TypeError` being raised.

    """

    def __init__(self, decorated):
        self._decorated = decorated

    def Instance(self):
        """
        Returns the singleton instance. Upon its first call, it creates
            ↪ a
        new instance of the decorated class and calls its `__init__`
            ↪ method.
        On all subsequent calls, the already created instance is
            ↪ returned.

        """
        try:
            return self._instance
        except AttributeError:
            self._instance = self._decorated()
            return self._instance

    def __call__(self):
        raise TypeError('Singletons must be accessed through `Instance()
            ↪ `.')

    def __instancecheck__(self, inst):
        return isinstance(inst, self._decorated)
```

Thread safe

werediver's implementation of singletons. A thread safe implementation of singleton pattern in Python. Based on tornado.ioloop.IOLoop.instance() approach.
import threading
Based on tornado.ioloop.IOLoop.instance() approach. See https://github.com/facebook/tornado
class SingletonMixin(object): $_{singleton_lock=threading.Lock()_singleton_instance=None}$
@classmethod def instance(cls): if not cls.$_{singleton_instance:withcls._singleton_lock:ifnotcls._singleton_instance:cls._singleton_instan}$

class A(SingletonMixin): pass
class B(SingletonMixin): pass
if $_{name=='main':a,a2=A.instance(),A.instance()b,b2=B.instance(),B.instance()}$
assert a is a2 assert b is b2 assert a is not b
print('a: print('b: Suggested Readings Is there a simple, elegant way to define singletons?

1.8 File System & IO

1.8.1 JSON

Write json file with pretty format and unicode

```
import json
import io

data = {
    "menu": {
        "header": "Sample Menu",
        "items": [
            {"id": "Open"},
            {"id": "OpenNew", "label": "Open New"},
            None,
            {"id": "Help"},
            {"id": "About", "label": "About Adobe CVG Viewer..."}
        ]
    }}

with io.open("sample_json.json", "w", encoding="utf8") as f:
    content = json.dumps(data, indent=4, sort_keys=True, ensure_ascii
        ↪ =False)
    f.write(unicode(content))
```

Output

```
{
    "menu": {
        "header": "Sample Menu",
        "items": [
            {
                "id": "Open"
            },
            {
                "id": "OpenNew",
```

```
                "label": "Open New"
            },
            null,
            {
                "id": "Help"
            },
            {
                "id": "About",
                "label": "About Adobe CVG Viewer..."
            }
        ]
    }
}
```

Read json file

```
import json
from pprint import pprint

with open('sample_json.json') as data_file:
    data = json.load(data_file)

pprint(data)
```

Output

```
{u'menu': {u'header': u'Sample Menu',
           u'items': [{u'id': u'Open'},
                      {u'id': u'OpenNew', u'label': u'Open New'},
                      None,
                      {u'id': u'Help'},
                      {u'id': u'About',
                       u'label': u'About Adobe CVG Viewer...'}]}}
```

Related Reading
Parsing values from a JSON file in Python, stackoverflow How do I write JSON data to a file in Python?, stackoverflow

1.8.2 XML

Write xml file with lxml package

```
import lxml.etree as ET
# root declaration
root = ET.Element('catalog')
# insert comment
comment = ET.Comment(' this is a xml sample file ')
root.insert(1, comment)
# book element
book = ET.SubElement(root, 'book', id="bk001")
# book data
author = ET.SubElement(book, 'author')
author.text = "Gambardella, Matthew"
```

```
title  = ET.SubElement(book, 'title')
title .text  = "XML Developer's Guide"
# write xml to  file
tree  = ET.ElementTree(root)
tree.write("sample_book.xml", pretty_print=True, xml_declaration=
    ↪ True, encoding='utf-8')
```

Output

```
<?xml version='1.0' encoding='UTF-8'?>
<catalog>
  <!-- this is a xml sample file -->
  <book id="bk001">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
  </book>
</catalog>
```

Read xml file with lxml package

```
from lxml import etree as ET

tree  = ET.parse("sample_book.xml")
root  = tree.getroot()
book = root.find('book')
print "Book Information"
print "ID      :", book.attrib["id"]
print "Author :", book.find('author').text
print "Title   :", book.find(' title ').text
```

Output

```
Book Information
ID      : bk001
Author : Gambardella, Matthew
Title   : XML Developer's Guide
```

1.9 Khóa học 2: Lập trình Python nâng cao

1.9.1 Mục tiêu của khoá học

Tìm hiểu các vấn đề lập trình Python nâng cao qua các ví dụ thực tế, sinh động

1.9.2 Đối tượng học viên

- Là sinh viên năm 2, năm 3
- Đang học các môn Lập trình song song, phát triển Web

1.10 Yield and Generators

Coroutines and Subroutines When we call a normal Python function, execution starts at function's first line and continues until a return statement, exception, or

the end of the function (which is seen as an implicit return None) is encountered. Once a function returns control to its caller, that's it. Any work done by the function and stored in local variables is lost. A new call to the function creates everything from scratch.
This is all very standard when discussing functions (more generally referred to as subroutines) in computer programming. There are times, though, when it's beneficial to have the ability to create a "function" which, instead of simply returning a single value, is able to yield a series of values. To do so, such a function would need to be able to "save its work," so to speak.
I said, "yield a series of values" because our hypothetical function doesn't "return" in the normal sense. return implies that the function is returning control of execution to the point where the function was called. "Yield," however, implies that the transfer of control is temporary and voluntary, and our function expects to regain it in the future.
In Python, "functions" with these capabilities are called generators, and they're incredibly useful. generators (and the yield statement) were initially introduced to give programmers a more straightforward way to write code responsible for producing a series of values. Previously, creating something like a random number generator required a class or module that both generated values and kept track of state between calls. With the introduction of generators, this became much simpler.
To better understand the problem generators solve, let's take a look at an example. Throughout the example, keep in mind the core problem being solved: generating a series of values.
Note: Outside of Python, all but the simplest generators would be referred to as coroutines. I'll use the latter term later in the post. The important thing to remember is, in Python, everything described here as a coroutine is still a generator. Python formally defines the term generator; coroutine is used in discussion but has no formal definition in the language.
Example: Fun With Prime Numbers Suppose our boss asks us to write a function that takes a list of ints and returns some Iterable containing the elements which are prime1 numbers.
Remember, an Iterable is just an object capable of returning its members one at a time.
"Simple," we say, and we write the following:

```
def get_primes(input_list):
    result_list = list()
    for element in input_list:
        if is_prime(element):
            result_list.append()

    return result_list
```

or better yet...

```
def get_primes(input_list):
    return (element for element in input_list if is_prime(element))

# not germane to the example, but here's a possible implementation of
# is_prime...
```

```
def is_prime(number):
    if number > 1:
        if number == 2:
            return True
        if number % 2 == 0:
            return False
        for current in range(3, int(math.sqrt(number) + 1), 2):
            if number % current == 0:
                return False
        return True
    return False
```

Either $get_primesimplementationabovefulfillstherequirements, sowetellourbosswe'redone.Shereportsourf$
Dealing With Infinite Sequences Well, not quite exactly. A few days later, our boss comes back and tells us she's run into a small problem: she wants to use our $get_primesfunctiononaverylargelistofnumbers.Infact, thelistissolargethatmerelycreatingitwouldconsume$
Once we think about this new requirement, it becomes clear that it requires more than a simple change to $get_primes.Clearly, wecan'treturnalistofalltheprimenumbersfromstarttoinfinity($
Before we give up, let's determine the core obstacle preventing us from writing a function that satisfies our boss's new requirements. Thinking about it, we arrive at the following: functions only get one chance to return results, and thus must return all results at once. It seems pointless to make such an obvious statement; "functions just work that way," we think. The real value lies in asking, "but what if they didn't?"
Imagine what we could do if $get_primescouldsimplyreturnthenextvalueinsteadofallthevaluesatonce.Itwould$
Unfortunately, this doesn't seem possible. Even if we had a magical function that allowed us to iterate from n to infinity, we'd get stuck after returning the first value:
def $get_primes(start) : forelementinmagical_infinite_range(start) : ifis_prime(element) :$
$returnelementImagineget_primesiscalledlikeso :$
def $solve_number_10() : She * is * workingonProjectEuler10, Iknewit!total =$
$2fornext_primeinget_primes(3) : ifnext_prime < 2000000 : total+ = next_primeelse :$
$print(total)returnClearly, inget_primes, wewouldimmediatelyhitthecasewherenumber =$
$3andreturnatline4.Insteadofreturn, weneedawaytogenerateavalueand, whenaskedforthenextone, pickupw$
Functions, though, can't do this. When they return, they're done for good. Even if we could guarantee a function would be called again, we have no way of saying, "OK, now, instead of starting at the first line like we normally do, start up where we left off at line 4." Functions have a single entry point: the first line.
Enter the Generator This sort of problem is so common that a new construct was added to Python to solve it: the generator. A generator "generates" values. Creating generators was made as straightforward as possible through the concept of generator functions, introduced simultaneously.
A generator function is defined like a normal function, but whenever it needs to generate a value, it does so with the yield keyword rather than return. If the body of a def contains yield, the function automatically becomes a generator function (even if it also contains a return statement). There's nothing else we need to do to create one.
generator functions create generator iterators. That's the last time you'll see the term generator iterator, though, since they're almost always referred to as

"generators". Just remember that a generator is a special type of iterator. To be considered an iterator, generators must define a few methods, one of which is next(). To get the next value from a generator, we use the same built-in function as for iterators: next().
This point bears repeating: to get the next value from a generator, we use the same built-in function as for iterators: next().
(next() takes care of calling the generator's next() method). Since a generator is a type of iterator, it can be used in a for loop.
So whenever next() is called on a generator, the generator is responsible for passing back a value to whomever called next(). It does so by calling yield along with the value to be passed back (e.g. yield 7). The easiest way to remember what yield does is to think of it as return (plus a little magic) for generator functions.**
Again, this bears repeating: yield is just return (plus a little magic) for generator functions.
Here's a simple generator function:
»> def simple$_{g}enerator_{f}unction() :>>> yield1 >>> yield2 >>> yield3Andherearetwosimplewaystouseit$
»> for value in simple$_{g}enerator_{f}unction() :>>> print(value)123 >>> our_{g}enerator =$
$simple_{g}enerator_{f}unction() >>> next(our_{g}enerator)1 >>> next(our_{g}enerator)2 >>>$
$next(our_{g}enerator)3Magic?What'sthemagicpart?Gladyouasked!Whenagenerator functioncallsyield, the"$
Let's rewrite get$_{p}rimesasageneratorfunction.Noticethatwenolongerneedthemagical_{i}nfinite_{r}angefunction$
def get$_{p}rimes(number) : whileTrue : ifis_{p}rime(number) : yieldnumbernumber+ =$
$1Ifagenerator functioncallsreturnorreachestheenditsdefinition, aStopIterationexceptionisraised.Thissig$
$loopispresentinget_{p}rimes.Ifitweren't, thefirsttimenext()wascalledwewouldcheckifthenumberisprimeand$
»> our$_{g}enerator = simple_{g}enerator_{f}unction() >>> forvalueinour_{g}enerator :>>>$
$print(value)$
»> our$_{g}eneratorhasbeenexhausted... >>> print(next(our_{g}enerator))Traceback(mostrecentcalllast) :$
$File" < ipython - input - 13 - 7e48a609051a > ", line1, in < module >$
$next(our_{g}enerator)StopIteration$
»> however, we can always create a new generator »> by calling the generator function again...
»> new$_{g}enerator = simple_{g}enerator_{f}unction() >>> print(next(new_{g}enerator))perfectlyvalid1Thus, theu$
Visualizing the flow Let's go back to the code that was calling get$_{p}rimes$: $solve_{n}umber_{1}0$.
def solve$_{n}umber_{1}0() : She * is * workingonProjectEuler10, Iknewit!total =$
$2fornext_{p}rimeinget_{p}rimes(3) : ifnext_{p}rime < 2000000 : total+ = next_{p}rimeelse :$
$print(total)returnIt'shelpfultovisualizehowthefirstfewelementsarecreatedwhenwecallget_{p}rimesinsolve_{n}$
We enter the while loop on line 3 The if condition holds (3 is prime) We yield the value 3 and control to solve$_{n}umber_{1}0.Then, backinsolve_{n}umber_{1}0$:
The value 3 is passed back to the for loop The for loop assigns next$_{p}rimetothisvaluenext_{p}rimeisaddedtototalT$
def get$_{p}rimes(number) : whileTrue : ifis_{p}rime(number) : yieldnumbernumber+ =$
$1 <<<<<<<<<< Mostimportantly, numberstillhasthesamevalueitdidwhenwecalledyield(i.e.3).Rememb$
Moar Power In PEP 342, support was added for passing values into generators. PEP 342 gave generators the power to yield a value (as before), receive a value, or both yield a value and receive a (possibly different) value in a single statement. To illustrate how values are sent to a generator, let's return to our prime number example. This time, instead of simply printing every prime number greater than number, we'll find the smallest prime number greater than successive powers of a number (i.e. for 10, we want the smallest prime greater than 10, then 100, then 1000, etc.). We start in the same way as get$_{p}rimes$:

def print$_successive_primes(iterations, base = 10) : likenormalfunctions, ageneratorfunctioncanbeassigned$
prime$_generator = get_primes(base)missingcode...forpowerinrange(iterations) :$
$missingcode...$
def get$_primes(number) : whileTrue : ifis_prime(number) : ...whatgoeshere?Thenextlineofget_primestakesa$
$yieldfoomeans, "yieldfooand, whenavalueissenttome, setothertothatvalue."Youcan"send"valuestoagenerat$
def get$_primes(number) : whileTrue : ifis_prime(number) : number = yieldnumbernumber+ =$
$1Inthisway, wecansetnumbertoadifferentvalueeachtimethegeneratoryields.Wecannowfillinthemissingco$
def print$_successive_primes(iterations, base = 10) : prime_generator = get_primes(base)prime_generator.send($
$print(prime_generator.send(base**power))Twothingstonotehere : First, we'reprintingtheresultofgenerato$
Second, notice the prime$_generator.send(None)line.Whenyou'reusingsendto"start"agenerator(thatis, execu$
Round-up In the second half of this series, we'll discuss the various ways in which generators have been enhanced and the power they gained as a result. yield has become one of the most powerful keywords in Python. Now that we've built a solid understanding of how yield works, we have the knowledge necessary to understand some of the more "mind-bending" things that yield can be used for.

Believe it or not, we've barely scratched the surface of the power of yield. For example, while send does work as described above, it's almost never used when generating simple sequences like our example. Below, I've pasted a small demonstration of one common way send is used. I'll not say any more about it as figuring out how and why it works will be a good warm-up for part two.

```
import random

def get_data():
    """Return 3 random integers between 0 and 9"""
    return random.sample(range(10), 3)

def consume():
    """Displays a running average across lists of integers sent to it"""
    running_sum = 0
    data_items_seen = 0

    while True:
        data = yield
        data_items_seen += len(data)
        running_sum += sum(data)
        print('The running average is {}'.format(running_sum / float(
            ↪ data_items_seen)))

def produce(consumer):
    """Produces a set of values and forwards them to the pre-defined
        ↪ consumer
    function"""
    while True:
        data = get_data()
        print('Produced {}'.format(data))
        consumer.send(data)
        yield
```

```
if __name__ == '__main__':
    consumer = consume()
    consumer.send(None)
    producer = produce(consumer)

    for _ in range(10):
        print('Producing...')
        next(producer)
```

Remember... There are a few key ideas I hope you take away from this discussion: generators are used to generate a series of values yield is like the return of generator functions The only other thing yield does is save the "state" of a generator function A generator is just a special type of iterator Like iterators, we can get the next value from a generator using next() for gets values by calling next() implicitly

1.10.1 Metaclasses

Metaclasses Python, Classes, and Objects Most readers are aware that Python is an object-oriented language. By object-oriented, we mean that Python can define classes, which bundle data and functionality into one entity. For example, we may create a class IntContainer which stores an integer and allows certain operations to be performed:

```
class IntContainer(object):
    def __init__(self, i):
        self.i = int(i)

    def add_one(self):
        self.i += 1
ic = IntContainer(2)
ic.add_one()
print(ic.i)
3
```

This is a bit of a silly example, but shows the fundamental nature of classes: their ability to bundle data and operations into a single object, which leads to cleaner, more manageable, and more adaptable code. Additionally, classes can inherit properties from parents and add or specialize attributes and methods. This object-oriented approach to programming can be very intuitive and powerful.
What many do not realize, though, is that quite literally everything in the Python language is an object.
For example, integers are simply instances of the built-in int type:
print type(1) <type 'int'> To emphasize that the int type really is an object, let's derive from it and specialize the $_{a}dd_{m}ethod(whichisthemachineryunderneaththe+operator)$:
(Note: We'll used the super syntax to call methods from the parent class: if you're unfamiliar with this, take a look at this StackOverflow question).

```
class MyInt(int):
    def __add__(self, other):
        print "specializing addition"
        return super(MyInt, self).__add__(other)
```

```
i = MyInt(2)
print(i + 2)
specializing addition
4
```

Using the + operator on our derived type goes through our $_{add_{method,asexpected.Weseethatintreallyisanobjectthatcanbesubcl}}$
Down the Rabbit Hole: Classes as Objects We said above that everything in python is an object: it turns out that this is true of classes themselves. Let's look at an example.
We'll start by defining a class that does nothing
class DoNothing(object): pass If we instantiate this, we can use the type operator to see the type of object that it is:
d = DoNothing() type(d) $_{main.DoNothingWeseethatourvariabledisaninstanceoftheclass_{main.DoNothing.}}$
We can do this similarly for built-in types:
L = [1, 2, 3] type(L) list A list is, as you may expect, an object of type list.
But let's take this a step further: what is the type of DoNothing itself?
type(DoNothing) type The type of DoNothing is type. This tells us that the class DoNothing is itself an object, and that object is of type type.
It turns out that this is the same for built-in datatypes:
type(tuple), type(list), type(int), type(float) (type, type, type, type) What this shows is that in Python, classes are objects, and they are objects of type type. type is a metaclass: a class which instantiates classes. All new-style classes in Python are instances of the type metaclass, including type itself:
type(type) type Yes, you read that correctly: the type of type is type. In other words, type is an instance of itself. This sort of circularity cannot (to my knowledge) be duplicated in pure Python, and the behavior is created through a bit of a hack at the implementation level of Python.
Metaprogramming: Creating Classes on the Fly Now that we've stepped back and considered the fact that classes in Python are simply objects like everything else, we can think about what is known as metaprogramming. You're probably used to creating functions which return objects. We can think of these functions as an object factory: they take some arguments, create an object, and return it. Here is a simple example of a function which creates an int object:
def $int_{factory}(s) : i = int(s)returni$
i = $int_{factory}('100')print(i)100Thisisoverly-simplistic, butanyfunctionyouwriteinthecourseofanormalpr$
$takesomearguments, dosomeoperations, andcreatereturnanobject.Withtheabovediscussioninmind, though,$
$-thisisametafunction :$
def $class_{factory}() : classFoo(object) : passreturnFoo$
F = $class_{factory}()f = F()print(type(f)) < class'_{main.Foo'>Justasthefunctionint_{factory}constructsanreturnsaninstanceofint}$
But the above construction is a bit awkward: especially if we were going to do some more complicated logic when constructing Foo, it would be nice to avoid all the nested indentations and define the class in a more dynamic way. We can accomplish this by instantiating Foo from type directly:
def $class_{factory}() : returntype('Foo', (),)$
F = $class_{factory}()f = F()print(type(f)) < class'_{main.Foo'>Infact,theconstruct}$
class MyClass(object): pass is identical to the construct
MyClass = type('MyClass', (),) MyClass is an instance of type type, and that can be seen explicitly in the second version of the definition. A potential confusion arises from the more common use of type as a function to determine

the type of an object, but you should strive to separate these two uses of the keyword in your mind: here type is a class (more precisely, a metaclass), and MyClass is an instance of type.
The arguments to the type constructor are: type(name, bases, dct) - name is a string giving the name of the class to be constructed - bases is a tuple giving the parent classes of the class to be constructed - dct is a dictionary of the attributes and methods of the class to be constructed
So, for example, the following two pieces of code have identical results:
class Foo(object): i = 4
class Bar(Foo): def get$_i(self)$: $returnself.i$
b = Bar() print(b.get$_i$())$4Foo = type('Foo', (), dict(i = 4))$
Bar = type('Bar', (Foo,), dict(get$_i$ = $lambdaself$: $self.i$))
b = Bar() print(b.get$_i$())$4Thisperhapsseemsabitover-complicatedinthecaseofthiscontrivedexample, butitc$
$the-fly.$
Making Things Interesting: Custom Metaclasses Now things get really fun. Just as we can inherit from and extend a class we've created, we can also inherit from and extend the type metaclass, and create custom behavior in our metaclass.
Example 1: Modifying Attributes Let's use a simple example where we want to create an API in which the user can create a set of interfaces which contain a file object. Each interface should have a unique string ID, and contain an open file object. The user could then write specialized methods to accomplish certain tasks. There are certainly good ways to do this without delving into metaclasses, but such a simple example will (hopefully) elucidate what's going on.
First we'll create our interface meta class, deriving from type:
class InterfaceMeta(type): def $_new_{(cls,name,parents,dct):createaclass_idifit'snotspecifiedif'class_id'notindct:dct['class_id']=name.lowe}$
open the specified file for writing if 'file' in dct: filename = dct['file'] dct['file'] = open(filename, 'w')
we need to call type.$_new_{tocompletetheinitializationreturnsuper(InterfaceMeta,cls)._new_{(cls,name,parents,dct)Noticethatwe'vemodifiedt}$
Now we'll use our InterfaceMeta class to construct and instantiate an Interface object:
Interface = InterfaceMeta('Interface', (), dict(file='tmp.txt'))
print(Interface.class$_id)print(Interface.file)interface < openfile'tmp.txt', mode'w'at0x21b8810 >$
$Thisbehavesaswe'dexpect : theclass_idclassvariableiscreated, andthefileclassvariableisreplacedwithanopen$
class Interface(object): $_metaclass_{=InterfaceMetafile='tmp.txt'}$
print(Interface.class$_id)print(Interface.file)interface < openfile'tmp.txt', mode'w'at0x21b8ae0 >$
$bydefiningthe_{metaclass_attributeoftheclass,we'vetoldtheclassthatitshouldbeconstructedusingInterfaceMetaratherthanusingtype.Tomake}$
type(Interface) $_main_{InterfaceMetaFurthermore,anyclassderivedfromInterfacewillnowbeconstructedusingthesamemetaclass:}$
class UserInterface(Interface): file = 'foo.txt'
print(UserInterface.file) print(UserInterface.class$_id) < openfile'foo.txt', mode'w'at0x21b8c00 >$
$userinterfaceThissimpleexampleshowshowmetaclassescanbeusedtocreatepowerfulandflexibleAPIsforpr$
Example 2: Registering Subclasses Another possible use of a metaclass is to automatically register all subclasses derived from a given base class. For example, you may have a basic interface to a database and wish for the user to be able to define their own interfaces, which are automatically stored in a master registry. You might proceed this way:
class DBInterfaceMeta(type): we use $_init_{ratherthan_new_herebecausewewanttomodifyattributesoftheclass*after*theyhavebeencreat}$
super(DBInterfaceMeta, cls).$_init_{(name,bases,dct)Ourmetaclasssimplyaddsaregistrydictionaryifit'snotalreadypresent,andaddsthenew}$
class DBInterface(object): $_metaclass_{=DBInterfaceMeta}$

print(DBInterface.registry) Now let's create some subclasses, and double-check that they're added to the registry:
class FirstInterface(DBInterface): pass
class SecondInterface(DBInterface): pass
class SecondInterfaceModified(SecondInterface): pass
print(DBInterface.registry) 'firstinterface': <class '$_{main.FirstInterface'>,'secondinterface':<class'_{main.SecondInterface'>,'s}$

Conclusion: When Should You Use Metaclasses? I've gone through some examples of what metaclasses are, and some ideas about how they might be used to create very powerful and flexible APIs. Although metaclasses are in the background of everything you do in Python, the average coder rarely has to think about them.
But the question remains: when should you think about using custom metaclasses in your project? It's a complicated question, but there's a quotation floating around the web that addresses it quite succinctly:
Metaclasses are deeper magic than 99% of users should ever worry about. If you wonder whether you need them, you don't (the people who actually need them know with certainty that they need them, and don't need an explanation about why).
Tim Peters
In a way, this is a very unsatisfying answer: it's a bit reminiscent of the wistful and cliched explanation of the border between attraction and love: "well, you just... know!"
But I think Tim is right: in general, I've found that most tasks in Python that can be accomplished through use of custom metaclasses can also be accomplished more cleanly and with more clarity by other means. As programmers, we should always be careful to avoid being clever for the sake of cleverness alone, though it is admittedly an ever-present temptation.
I personally spent six years doing science with Python, writing code nearly on a daily basis, before I found a problem for which metaclasses were the natural solution. And it turns out Tim was right:
I just knew.

1.11 Hệ điều hành

1.11.1 File Operations

Copy folder

```
import shutil
shutil.copyfile("src", "dst")
```

1.11.2 CLI

shutil - High-level file operations

1.12 Cơ sở dữ liệu (chưa xây dựng)

1.13 Giao diện (chưa xây dựng)

1.14 Lập trình mạng

REST JSON 1 2 GET

```
import requests
url = "http://localhost:8080/messages"
response = requests.get(url)
data = response.json()
```

POST

```
import requests
import json

url = "http://localhost:8080/messages"
data = {'sender': 'Alice', 'receiver': 'Bob', 'message': 'Hello!'}
headers = {
  'Content-type': 'application/json',
  'Accept': 'application/json'}
r = requests.post(url, data=json.dumps(data), headers=headers)
```

1.15 Lập trình song song

Running several threads is similar to running several different programs concurrently, but with the following benefits
Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes. Threads sometimes called light-weight processes and they do not require much memory overhead; they are cheaper than processes. A thread has a beginning, an execution sequence, and a conclusion. It has an instruction pointer that keeps track of where within its context it is currently running.
It can be pre-empted (interrupted)
It can temporarily be put on hold (also known as sleeping) while other threads are running - this is called yielding. Starting a New Thread To spawn another thread, you need to call following method available in thread module:
thread.start$_new_thread(function, args[, kwargs])This method call enables a fast and efficient way to create new$
The method call returns immediately and the child thread starts and calls function with the passed list of args. When function returns, the thread terminates. Here, args is a tuple of arguments; use an empty tuple to call function without passing any arguments. kwargs is an optional dictionary of keyword arguments. Example

```
#!/usr/bin/python

import thread
```

```
import time

# Define a function for the thread
def print_time( threadName, delay):
   count = 0
   while count < 5:
      time.sleep(delay)
      count += 1
      print "%s: %s" % ( threadName, time.ctime(time.time()) )

# Create two threads as follows
try:
   thread.start_new_thread( print_time, ("Thread-1", 2, ) )
   thread.start_new_thread( print_time, ("Thread-2", 4, ) )
except:
   print "Error: unable to start thread"

while 1:
   pass
```

When the above code is executed, it produces the following result

```
Thread-1: Thu Jan 22 15:42:17 2009
Thread-1: Thu Jan 22 15:42:19 2009
Thread-2: Thu Jan 22 15:42:19 2009
Thread-1: Thu Jan 22 15:42:21 2009
Thread-2: Thu Jan 22 15:42:23 2009
Thread-1: Thu Jan 22 15:42:23 2009
Thread-1: Thu Jan 22 15:42:25 2009
Thread-2: Thu Jan 22 15:42:27 2009
Thread-2: Thu Jan 22 15:42:31 2009
Thread-2: Thu Jan 22 15:42:35 2009
```

Although it is very effective for low-level threading, but the thread module is very limited compared to the newer threading module.

The Threading Module

The newer threading module included with Python 2.4 provides much more powerful, high-level support for threads than the thread module discussed in the previous section.

The threading module exposes all the methods of the thread module and provides some additional methods:

threading.activeCount(): Returns the number of thread objects that are active. threading.currentThread(): Returns the number of thread objects in the caller's thread control. threading.enumerate(): Returns a list of all thread objects that are currently active. In addition to the methods, the threading module has the Thread class that implements threading. The methods provided by the Thread class are as follows:

run(): The run() method is the entry point for a thread. start(): The start() method starts a thread by calling the run method. join([time]): The join() waits for threads to terminate. isAlive(): The isAlive() method checks whether a thread is still executing. getName(): The getName() method returns the name of a thread. setName(): The setName() method sets the name of a thread.

Creating Thread Using Threading Module
To implement a new thread using the threading module, you have to do the following
Define a new subclass of the Thread class. Override the init(self [,args]) method to add additional arguments. Then, override the run(self [,args]) method to implement what the thread should do when started. Once you have created the new Thread subclass, you can create an instance of it and then start a new thread by invoking the start(), which in turn calls run() method.
Example

```
#!/usr/bin/python

import threading
import time

exitFlag = 0

class myThread (threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        print "Starting " + self.name
        print_time(self.name, self.counter, 5)
        print "Exiting " + self.name

def print_time(threadName, delay, counter):
    while counter:
        if exitFlag:
            threadName.exit()
        time.sleep(delay)
        print "%s: %s" % (threadName, time.ctime(time.time()))
        counter -= 1

# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# Start new Threads
thread1.start()
thread2.start()

print "Exiting Main Thread"
```

When the above code is executed, it produces the following result

```
Starting Thread-1
Starting Thread-2
Exiting Main Thread
```

```
Thread-1: Thu Mar 21 09:10:03 2013
Thread-1: Thu Mar 21 09:10:04 2013
Thread-2: Thu Mar 21 09:10:04 2013
Thread-1: Thu Mar 21 09:10:05 2013
Thread-1: Thu Mar 21 09:10:06 2013
Thread-2: Thu Mar 21 09:10:06 2013
Thread-1: Thu Mar 21 09:10:07 2013
Exiting Thread-1
Thread-2: Thu Mar 21 09:10:08 2013
Thread-2: Thu Mar 21 09:10:10 2013
Thread-2: Thu Mar 21 09:10:12 2013
Exiting Thread-2
```

Synchronizing Threads

The threading module provided with Python includes a simple-to-implement locking mechanism that allows you to synchronize threads. A new lock is created by calling the Lock() method, which returns the new lock.

The acquire(blocking) method of the new lock object is used to force threads to run synchronously. The optional blocking parameter enables you to control whether the thread waits to acquire the lock.

If blocking is set to 0, the thread returns immediately with a 0 value if the lock cannot be acquired and with a 1 if the lock was acquired. If blocking is set to 1, the thread blocks and wait for the lock to be released.

The release() method of the new lock object is used to release the lock when it is no longer required.

Example

```
#!/usr/bin/python

import threading
import time

class myThread (threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        print "Starting " + self.name
        # Get lock to synchronize threads
        threadLock.acquire()
        print_time(self.name, self.counter, 3)
        # Free lock to release next thread
        threadLock.release()

def print_time(threadName, delay, counter):
    while counter:
        time.sleep(delay)
        print "%s: %s" % (threadName, time.ctime(time.time()))
        counter -= 1
```

```
threadLock = threading.Lock()
threads = []

# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# Start new Threads
thread1.start()
thread2.start()

# Add threads to thread list
threads.append(thread1)
threads.append(thread2)

# Wait for all threads to complete
for t in threads:
    t.join()
print "Exiting Main Thread"
```

When the above code is executed, it produces the following result

```
Starting Thread-1
Starting Thread-2
Starting Thread-3
Thread-1 processing One
Thread-2 processing Two
Thread-3 processing Three
Thread-1 processing Four
Thread-2 processing Five
Exiting Thread-3
Exiting Thread-1
Exiting Thread-2
Exiting Main Thread
```

Related Readings "Python Multithreaded Programming". www.tutorialspoint.com. N.p., 2016. Web. 13 Dec. 2016. "An Introduction To Python Concurrency". dabeaz.com. N.p., 2016. Web. 14 Dec. 2016.

1.16 Event Based Programming

Introduction: pydispatcher 1 2
PyDispatcher provides the Python programmer with a multiple-producer-multiple-consumer signal-registration and routing infrastructure for use in multiple contexts. The mechanism of PyDispatcher started life as a highly rated recipe in the Python Cookbook. The project aims to include various enhancements to the recipe developed during use in various applications. It is primarily maintained by Mike Fletcher. A derivative of the project provides the Django web framework's "signal" system.
Used by Django community

Usage 1 To set up a function to receive signals: from pydispatch import dispatcher
SIGNAL = 'my-first-signal'
def handle$_event(sender)$: """$Simpleeventhandler"""print'Signalwassentby', sender$
dispatcher.connect(handle$_event, signal = SIGNAL, sender = dispatcher.Any$)
The use of the Any object allows the handler to listen for messages from any Sender or to listen to Any message being sent. To send messages: first$_sender =$
$object()second_sender =$
def main(): dispatcher.send(signal=SIGNAL, sender=first$_sender)dispatcher.send(signal =$
$SIGNAL, sender = second_sender$)
Which causes the following to be printed:
Signal was sent by <object object at 0x196a090> Signal was sent by Messaging Conda link Docker link Github - pubSubService Github - pubSubClient Pypi link
Python Publish - Subscribe Pattern Implementation:
Step by Step to run PubSub: Step 1: Pull pubsub image from docker hub run it: docker pull hunguyen/pubsub:latest docker run -d -p 8000:8000 hunguyen/pubsub Step 2: To run client first install pyconfiguration from conda conda install -c rain1024 pyconfiguration Step 3: Install pubSubClient package from conda conda install -c hunguyen pubsubclient Step 4: Create config.json file "PUBLISH$_SUBSCRIBE_SERVICE$" : "$http : //api.service.com"Step$5 :
$Runpubsubclientcreateandregisterorsyncapublisherpublisher = Publisher('P1')createanewtopictopic =
$Topic('A')createaneventofatopicevent = Event(topic)publisherpublishesaneventpublisher.publish(event)c
$Subscriber('S1')subscribersubscribestoatopicsubscriber.subscribe(topic)subscribergetallneweventsbytimest
$subscriber.get_events()pydispatcher$
stackoverflow, Recommended Python publish/subscribe/dispatch module?

1.17 Web Development

Django 1 Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.
Project Folder Structure
project$_folder/your_project_name/your_project_name/static/models.pyserializers.pysettings.pyurls.pyviews.p$
$InstalldependenciespipinstalldjangopipinstalldjangorestframeworkpipinstallmarkdownMarkdownsuppo$
$filterFilteringsupportpipinstalldjango - cors - headersCORSsupportStep2 :$
$Createprojectdjango−adminstartprojectyour_projectnameStep3 : Configapps3Add'your_projectname',' res$
INSTALLED$_APPS = (...'your_projectname''rest_framework',)Step4 : Model, View, Route6Step4.1 :$
$CreatemodelandserializerYoucangotoDjango : Modelfieldreferencepageformorefields.$
Step 4.1.1: Create Task class in your$_projectname/models.pyfilefromdjango.dbimportmodels$
class Task(models.Model): content = models.CharField(max$_length = 30)status =$
$models.CharField(max_length = 30)Step4.1.2 : CreateTaskSerializerclassinyour_projectname/serializers$
class TaskSerializer(serializers.HyperlinkedModelSerializer): class Meta: model = Task fields = ('id', 'content', 'status') Step 4.1.3: Create table in database 4
python manage.py syncdb With django 1.9
python manage.py makemigrations your$_projectnamepythonmanage.pymigrateStep4.2 :$
$CreateTaskViewSetclassinyour_projectname/views.pyfilefromyour_projectname.modelsimportTaskfrom$

class TaskViewSet(viewsets.ModelViewSet): queryset = Task.objects.all() serializer$_class$ = $TaskSerializerStep4.3 : Configroute5Changeyour_projectname/urls.pyfile$
from django.conf.urls import include, url from django.contrib import admin
from rest$_framework$importroutersfromyour$_projectname.viewsimportTaskViewSet$
router = routers.DefaultRouter() router.register(r'api/tasks', TaskViewSet) admin.autodiscover()
urlpatterns = [url(r'$^admin/', include(admin.site.urls)), url(r'^', include(router.urls)), url(r'^api-auth/', include('rest_framework.urls', namespace =' rest_framework'))]$Step5 : RunServerpythonmanage.pyrunserverStep6.UseAPIStep6.1 : Createanewtaskcurl-i - XPOST - H"Content - Type : application/json"http : //localhost : 8000/api/tasks-d'"content" : "a","status" : "INIT"'Step6.2 : Listalltaskscurlhttp : //localhost : 8000/api/tasksStep6.3 : Getdetailoftask1curlhttp : //localhost : 8000/api/tasks/1Step6.4 : Deletetask1curl-i-XDELETEhttp : //localhost : 8000/api/tasks/1Step7 : CORSKnownError : No'Access-Control-Allow-Origin'headerispresentontherequestedresource.Origin'null'istherefornotallowedaccess.$
Step 7.1: Install corsheader app Add module corsheaders to your$_projectname/settings.py$
INSTALLED$_APPS = (...'corsheaders', ...)Step7.2AddmiddlewareclassesAddmiddleware_classestoyour_proj$
MIDDLEWARE$_CLASSES = (...'corsheaders.middleware.CorsMiddleware',' django.middleware.common$
AllowAll
Add this line to your$_projectname/settings.py$
CORS$_ORIGIN_ALLOW_ALL : TrueStep8 : httpsYoucanusehttps : //github.com/teddziuba/django-sslserver$
Unicode REST$_FRAMEWORK = 'DEFAULT_RENDERER_CLASSES' : ('rest_framework.renderers.JS$
$PagingAddthismodulesettingtoyour_projectname/settings.py$
REST$_FRAMEWORK = 'DEFAULT_PAGINATION_CLASS' :' rest_framework.pagination.LimitOffset$
API:
GET <>/?limit=<limit>offset=<offset>
Step 10: Search by field in import this to your viewsets.py
from rest$_framework$importfilters
add this to your viewsets class
filter$_backends = (filters.SearchFilter,)search_fields = ('< field >','< field >'$
,)
One-to-Many Relationship 7 from django.db import models
class User(models.Model): name = models.TextField()
def $_str_{(self):return"-".format(str(self.id),self.name)}$
class Task(models.Model): name = models.TextField() assign = models.ForeignKey(User,
on$_delete = models.CASCADE)StartingwithMysqlAddthisdatabasesettingstoyour_projectname/settings.p$
DATABASES = 'default': 'ENGINE': 'django.db.backends.mysql', 'NAME':
'[DB$_NAME]','USER' :' [DB_USER]','PASSWORD' :' [PASSWORD]','HOST' :' [HOST]', OranIPAddressthatyourDBishostedon'PORT' :' 3306',$
Install this module to your virtual environment
conda install mysql-python if you are using virtual environment
pip install mysql-python if you using are root environment
Custom View 8 from rest$_frameworkimportmixins$
class CreateModelMixin(object): """ Create a model instance. """ def create(self, request, *args, **kwargs): event = request.data try: event['time'] = int(time.time()) except Exception, e: print 'Set Time Error' serializer = self.get$_serializer(data = request.data)serializer.is_valid(raise_exception = True)self.perform_create(serializer)headers =$

$self.get_success_headers(serializer.data) return Response(serializer.data, status = status.HTTP_201_CREATED, headers = headers)$
def perform$_create(self, serializer) : serializer.save()$
def get$_success_headers(self, data) : try : return 'Location' : data[api_settings.URL_FIELD_NAME] except(Ty$
$return$
class YourViewSet(CreateModelMixin, mixins.RetrieveModelMixin, mixins.UpdateModelMixin, mixins.DestroyModelMixin, mixins.ListModelMixin, GenericViewSet): queryset = YourModel.objects.all() serializer$_class = YourModelSerializer Loggingsettings Hereisanexample, putthi$
LOGGING = 'version': 1, 'disable$_existing_loggers' : False,' formatters' : 'verbose' : 'format' :',' simple' : 'format' :',' filters' : 'special' : '()' : ' project.logging.SpecialFilter',' foo' 'console' : 'level' : ' INFO',' filters' : ['require$_debug_true'],' class' : ' logging.StreamHandler',' formatter' :' 'django' : 'handlers' : ['console'],' propagate' : True,,' django.request' : 'handlers' : ['mail$_admins'],' level' :'$
Python: Build Python API Client package Step 1: Write document on Swagger Editor1 Step 2: Genenrate Client –> Python –> save python-client.zip Step 3: Extract zip Step 4: Open project in Pycharm rename project directory, project name, swagger$_client package Step 5 : 2 mkdir conda cd conda git clone https : //github.com/hunguyen1702/conda$
$rf.git README.md Step 6 : Edit meta.yaml file in your_package folder 6.1 Follow instruction inside meta.yaml$
$build : -python - setuptools run : -python with : requirements : build :$
$-python - setuptools - six - certifi - python - dateutil run : -python -$
$six - certifi - python - dateutil Step 7 : cd..conda build your_package Step 8 :$
$mkdir channel cd channel conda convert --platform all /anaconda/conda-bld/linux-$
$64/your_package_0.1.0-py27_0.tar.bz2 Step 9 : Create virtual-env name : your_env_name dependencies :$
$-certifi = 2016.2.28 = py27_0 - openssl = 1.0.2h = 0 - pip = 8.1.2 =$
$py27_0 - python = 2.7.11 = 0 - python - dateutil = 2.5.3 = py27_0 - readline =$
$6.2 = 2 - setuptools = 20.7.0 = py27_0 - six = 1.10.0 = py27_0 - tk = 8.5.18 =$
$0 - wheel = 0.29.0 = py27_0 - zlib = 1.2.8 = 0 - pip : -urllib3 == 1.15.1 Step 10 :$
$Install : conda install - -use - local your_package Django$
Writing your first Django app, part 1
Django REST framework: Installation
Django: Migrations
Building a Simple REST API for Mobile Applications
Django: Models
How to show object details in Django Rest Framework browseable API?
$rest_framework : mixins$

1.18 Khóa học 3: Phát triển phần mềm với Python

1.18.1 Mục tiêu của khóa học

1.18.2 Đối tượng học viên

- Đã lập trình Python được 1-2 năm
- Muốn phát triển phần mềm mã nguồn mở

1.19 Logging

levels, attributes references

The logging library takes a modular approach and offers several categories of components: loggers, handlers, filters, and formatters.
Loggers expose the interface that application code directly uses. Handlers send the log records (created by loggers) to the appropriate destination. Filters provide a finer grained facility for determining which log records to output. Formatters specify the layout of log records in the final output. Step 0: Project structure

```
code/
        main.py
        config
            logging.conf
        logs
            app.log
```

Step 1: Create file logging.conf
[loggers] keys=root
[handlers] keys=consoleHandler,fileHandler
[formatters] keys=formatter
[logger$_{r}oot]level = DEBUGhandlers = consoleHandler, fileHandler$
[handler$_{c}onsoleHandler]class = StreamHandlerlevel = DEBUGformatter = formatterargs = (sys.stdout,)$
[handler$_{f}ileHandler]class = FileHandlerlevel = DEBUGformatter = formatterargs = ('logs/app.log',' a')$
[formatter$_{f}ormatter]format = datefmt = Step2 : Loadconfigandcreatelogger$
In main.py
import logging.config
load logging config logging.config.fileConfig('config/logging.conf') Step 3: In your application code
logging.getLogger().debug('debug message') logging.getLogger().info('info message') logging.getLogger().warn('warn message') logging.getLogger().error('error message') logging.getLogger().critical('critical message') More Resources
Introduction to Logging Quick and simple usage of python log Python: Logging module
Python: Logging cookbook
Python: Logging guide

1.20 Configuration

pyconfiguration
Installation conda install -c rain1024 pyconfiguration Usage Step 1: Create config.json file
"SERVICE$_{U}RL$" : "$http : //api.service.com$" $Step2 : Addthesecodetomain.pyfile$
from pyconfiguration import Configuration Configuration.load('config.json') print Configuration.SERVICE$_{U}RL$
> http://api.service.com References: What's the best practice using a settings file 1
What's the best practice using a settings file in Python?

1.21 Command Line

Command Line Arguments There are the following modules in the standard library:
The getopt module is similar to GNU getopt. The optparse module offers object-oriented command line option parsing. Here is an example that uses the latter from the docs:
from optparse import OptionParser
parser = OptionParser() parser.add$_{o}ption$("$-f$","$--file$", $dest$ = "$filename$", $help$ = "$writereporttoFILE$", $metavar$ = "$FILE$")$parser.add_{o}ption$("$-q$","$--quiet$", $action$ = "$store_{f}alse$", $dest$ = "$verbose$", $default$ = $True$, $help$ = "$don'tprintstatusmessagestostdout$")
(options, args) = parser.parse$_{a}rgs$()$optparsesupports$($amongotherthings$)$:
Multiple options in any order. Short and long options. Default values. Generation of a usage help message. Suggest Reading Command Line Arguments In Python

1.22 Testing

Testing your code is very important.
Getting used to writing testing code and running this code in parallel is now considered a good habit. Used wisely, this method helps you define more precisely your code's intent and have a more decoupled architecture.
Unittest unittest is the batteries-included test module in the Python standard library. Its API will be familiar to anyone who has used any of the JUnit/-nUnit/CppUnit series of tools.
The Basics Creating test cases is accomplished by subclassing unittest.TestCase.
import unittest
def fun(x): return x + 1
class MyTest(unittest.TestCase): def test(self): self.assertEqual(fun(3), 4) Skipping tests Unittest supports skipping individual test methods and even whole classes of tests. In addition, it supports marking a test as an "expected failure," a test that is broken and will fail, but shouldn't be counted as a failure on a .code TestResult.
Skipping a test is simply a matter of using the skip() decorator or one of its conditional variants.
import sys import unittest
class MyTestCase(unittest.TestCase):
@unittest.skip("demonstrating skipping") def test$_{n}othing$($self$) : $self.fail$("$shouldn'thappen$")
@unittest.skipIf(mylib.$_{v}ersion_{<(1,3),"notsupportedinthislibraryversion")deftest_{f}ormat(self):Teststhatworkforonlyacertainversionofl$
@unittest.skipUnless(sys.platform.startswith("win"), "requires Windows") def test$_{w}indows_{s}upport$($self$) : $windowsspecifictestingcodepassToxtoxaimstoautomateandstandardizetesting$
Tox is a generic virtualenv management and test command line tool you can use for:
checking your package installs correctly with different Python versions and interpreters running your tests in each of the environments, configuring your test tool of choice acting as a frontend to Continuous Integration servers, greatly reducing boilerplate and merging CI and shell-based testing. Installation
You can install tox with pip using the following command

> pip install tox

Setup default environment in Windows with conda

> conda create −p C:\python27 python=2.7
> conda create −p C:\python34 python=3.4

Related Readings Testing Your Code, The Hitchhiker's Guide to Python unittest Unit testing framework, docs.python.org Is it possible to use tox with conda-based Python installations?, stackoverflow

1.23 IDE & Debugging

Today, I write some notes about my favorite Python IDE - PyCharm. I believe it's a good one for developing python, which supports git, vim, etc. This list below contains my favorite features.
Pycharm Features Intelligent Editor Navigation Graphical Debugger Refactorings Code Inspections Version Control Integration Scientific Tools Intelligent Editor PyCharm provides smart code completion, code inspections, on-the-fly error highlighting and quick-fixes, along with automated code refactorings and rich navigation capabilities.
Syntax Highlighting
Read your code easier with customizable colors for Python code and Django templates. Choose from several predefined color themes.
Auto-Identation and code formating
Automatic indents are inserted on new line. Indent verification and code reformatting are compliant with project code-style settings.
Configurable code styles
Select a predefined coding style to apply to your code style configuration for various supported languages.
Code completion
Code completion for keywords, classes, variables, etc. as you type or via Ctrl+Space. Editor suggestions are context-aware and offer the most appropriate options.
Keyboard shortcuts: Tab, Alt+Enter
Code selection and comments
Select a block of code and expand it to an expression, to a line, to a logical block of code, and so on with shortcuts. Single keystroke to comment/uncomment the current line or selection.
Code formatter
Code formatter with code style configuration and other features help you write neat code that's easy to support. PyCharm contains built-in PEP-8 for Python and other standards compliant code formatting for supported languages.
Code snippets and templates
Save time using advanced customizable and parametrized live code templates and snippets.
Keyboard shortcuts check.if ENTER
if check: type$_{s}omethingCodefolding$
Code folding, auto-insertion of braces, brackets quotes, matching brace/bracket highlighting, etc.
On-the-fly error highlighting
Errors are shown as you type. The integrated spell-checker verifies your identifiers and comments for misspellings.

Multiple carets and selections
With multiple carets, you can edit several locations in your file at the same time.
Keyboard shortcuts: SHIFT + F6
Code analysis
Numerous code inspections verify Python code as you type and also allow inspecting the whole project for possible errors or code smells.
Quick-fixes
Quick-fixes for most inspections make it easy to fix or improve the code instantly. Alt+Enter shows appropriate options for each inspection.
Keyboard shortcuts: F2
Duplicated code detector
Smart duplicated code detector analyzes your code and searches for copy/pasted code. You'll be presented with a list of candidates for refactoring and with the help of refactorings it's easy to keep your code dry.
Configurable language injections
Natively edit non-Python code embedded into string literals, with code completion, error-highlighting, and other coding assistance features.
Code auto generation
Code auto-generation from usage with quick-fixes; docstrings and the code matching verification, plus autoupdate on refactoring. Automatic generation of a docstring stub (reStructuredText, Epytext, Google, and NumPy).
Intention actions
Intention actions help you apply automated changes to code that is correct, to improve it or to make your coding routine easier.
Searching
Keyboard shortcuts: Double Shift (search everywhere)
Navigation Shortcuts
Keyboard shortcuts: ALT + SHIFT + UP/DOWN (move line up and down)
Graphical Debugger PyCharm provides extensive options for debugging your Python/Django and JavaScript code:
Set breakpoints right inside the editor and define hit conditions Inspect context-relevant local variables and user-defined watches, including arrays and complex objects, and edit values on the fly Set up remote debugging using remote interpreters Evaluate an expression in runtime and collect run-time type statistics for better autocompletion and code inspections Attach to a running process Debug Django templates
Inline Debugger
With an inline debugger, all live debugging data are shown directly in the editor, with variable values integrated into the editor's look-and-feel. Variable values can be viewed in the source code, right next to their usages.
Step into My Code
Use Step into My Code to stay focused on your code: the debugger will only step through your code bypassing any library sources.
Multi-process debugging
PyCharm can debug applications that spawn multiple Python processes, such as Django applications that don't run in –no-reload mode, or applications using many other Web frameworks that use a similar approach to code auto-reloading.
Run/Debug configurations

Every script/test or debugger execution creates a special 'Run/Debug Configuration' that can be edited and used later. Run/Debug Configurations can be shared with project settings for use by the whole team.
Workspace Custom Scheme Go to File - Settings... then Editor - Colors Fonts
Now you can change your scheme, I like Darcular
https://confluence.jetbrains.com/download/attachments/51945983/appearance3.png?version=1modification
IPython Support PyCharm supports usage of IPython magic commands.
http://i.stack.imgur.com/aTEW2.png
Vim Support You can configure PyCharm to work as a Vim editor
https://confluence.jetbrains.com/download/attachments/51946537/vim4.png?version=1modificationDate=1
Keyboard Shortcuts: Ctrl+Shift+V (paste)

1.23.1 Pycharm Pycharm

01/2018: Pycharm là trình duyệt ưa thích của mình trong suốt 3 năm vừa rồi.

Hôm nay tự nhiên lại gặp lỗi không tự nhận unittest, không resolve được package import bởi relative path. Vụ không tự nhận unittest sửa bằng cách xóa file .idea là xong. Còn vụ không resolve được package import bởi relative path thì vẫn chịu rồi. Nhìn code cứ đỏ lòm khó chịu thật.

1.24 Package Manager

1.24.1 py2exe

py2exe is a Python Distutils extension which converts Python scripts into executable Windows programs, able to run without requiring a Python installation.Spice
Installation

```
# py2exe
conda install -c https://conda.anaconda.org/clinicalgraphics cg-py2exe
Build 1
python setup.py py2exe
# build PyQT
python setup.py py2exe --includes sip
```

Known Issues
Error: Microsoft Visual C++ 10.0 is required (Unable to find vcvarsall.bat) (link)
How to fix
Step 1: Install Visual Studio 2015
Step 2:

```
set VS100COMNTOOLS=\%VS140COMNTOOLS\%
```

1.24.2 Quản lý gói với Anaconda

Cài đặt package tại một branch của một project trên github

```
> pip install git+https://github.com/tangentlabs/django-oscar-paypal
    ↪ .git@issue/34/oscar-0.6#egg=django-oscar-paypal
```

Trích xuất danh sách package

```
> pip freeze > requirements.txt
```

Chạy ipython trong environment anaconda
Chạy đống lệnh này

```
conda install nb_conda
source activate my_env
python -m IPython kernelspec install-self --user
ipython notebook
```

Interactive programming với ipython
Trích xuất ipython ra slide (không hiểu sao default '–to slides' không work nữa, lại phải thêm tham số '–reveal-prefix' [2]

```
jupyter nbconvert "file.ipynb"
  --to slides
  --reveal-prefix "https://cdnjs.cloudflare.com/ajax/libs/reveal.js
    ↪ /3.1.0"
```

Tham khảo thêm
* https://stackoverflow.com/questions/37085665/in-which-conda-environment-is-jupyter-executing * https://github.com/jupyter/notebook/issues/541issuecomment-146387578 * https://stackoverflow.com/a/20101940/772391

1.25 Environment

Similar to pip, conda is an open source package and environment management system 1. Anaconda is a data science platform that comes with a lot of packages. It uses conda at the core. Unlike Anaconda, Miniconda doesn't come with any installed packages by default. Note that for miniconda, everytime you open up a terminal, conda won't automatically be available. Run the command below to use conda within miniconda.
Conda Let's first start by checking if conda is installed.

```
> conda --version

conda 4.2.12
To see the full documentation for any command, type the command
    ↪ followed by --help. For example, to learn about the conda
    ↪ update command:

> conda update --help
Once it has been confirmed that conda has been installed, we will now
    ↪ make sure that it is up to date.

> conda update conda

Using Anaconda Cloud api site https://api.anaconda.org
Fetching package metadata: ....
```

[2] https://github.com/jupyter/nbconvert/issues/91issuecomment-283736634

```
.Solving package specifications : .....

Package plan for installation in environment //anaconda:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
        ↪
    conda-env-2.6.0            |                0         601 B
    ruamel_yaml-0.11.14        |          py27_0         184 KB
    conda-4.2.12               |          py27_0         376 KB
    ------------------------------------------------------------
        ↪
                                           Total:         560 KB

The following NEW packages will be INSTALLED:

    ruamel_yaml: 0.11.14-py27_0

The following packages will be UPDATED:

    conda:       4.0.7-py27_0 --> 4.2.12-py27_0
    conda-env:   2.4.5-py27_0 --> 2.6.0-0
    python:      2.7.11-0     --> 2.7.12-1
    sqlite :     3.9.2-0      --> 3.13.0-0

Proceed ([y]/n)? y

Fetching packages ...
conda-env-2.6. 100% |#
    ↪ ################################| Time:
    ↪ 0:00:00 360.78 kB/s
ruamel_yaml-0. 100% |#
    ↪ ################################| Time:
    ↪ 0:00:00 5.53 MB/s
conda-4.2.12-p 100% |#
    ↪ ################################| Time:
    ↪ 0:00:00 5.84 MB/s
Extracting packages ...
[       COMPLETE  ]|#
    ↪ ################################################|
    ↪  100%
Unlinking packages ...
[       COMPLETE  ]|#
    ↪ ################################################|
    ↪  100%
Linking packages ...
[       COMPLETE  ]|#
    ↪ ################################################|
```

↪ *100%*
Environments

1.25.1 Create

In order to manage environments, we need to create at least two so you can move or switch between them. To create a new environment, use the conda create command, followed by any name you wish to call it:

 # create new environment
conda create −n <your_environment> python=2.7.11

1.25.2 Clone

Make an exact copy of an environment by creating a clone of it. Here we will clone snowflakes to create an exact copy named flowers:

conda create −−name flowers −−clone snowflakes

1.25.3 List

List all environments
Now you can use conda to see which environments you have installed so far. Use the conda environment info command to find out

> conda info −e

conda environments:

snowflakes	/home/username/miniconda/envs/snowflakes
bunnies	/home/username/miniconda/envs/bunnies

Verify current environment
Which environment are you using right now - snowflakes or bunnies? To find out, type the command:

conda info −−envs

1.25.4 Remove

If you didn't really want an environment named flowers, just remove it as follows:

conda remove −−name flowers −−all

1.25.5 Share

You may want to share your environment with another person, for example, so they can re-create a test that you have done. To allow them to quickly reproduce your environment, with all of its packages and versions, you can give them a copy of your environment.yml file.
Export the environment file
To enable another person to create an exact copy of your environment, you will export the active environment file.

conda **env export** > environment.yml

Use environment from file
Create a copy of another developer's environment from their environment.yml file:

conda **env** create −f environment.yml
 # remove environment
conda remove −n <your_environemnt> −−all

1.26 Module

Create Public Module conda, pypi, github
Step 0/4: Check your package name Go to https://pypi.python.org/pypi/your$_{p}ackage_{n}ametoseeyourpackage$
Step 1/4: Make your module 1 1.1 pip install cookiecutter
1.2 cookiecutter https://github.com/audreyr/cookiecutter-pypackage.git
1.3 Fill all necessary information
full$_{n}ame[AudreyRoyGreenfeld]$: $email[aroy@alum.mit.edu]$: $github_{u}sername[audreyr]$: $project_{n}ame[PythonBoilerplate]$: $project_{s}lug[]$: $project_{s}hort_{d}escription$: $release_{d}ate[]$: $pypi_{u}sername[]$: $year[2016]$: $version[0.1.0]$: $use_{p}ypi_{d}eployment_{w}ith_{t}ravis[y]$: $Itwillcreateadirectory$
|- LICENSE |- README.md |- TODO.md |- docs | |– conf.py | |– generated | |– index.rst | |– installation.rst | |– modules.rst | |– quickstart.rst | |– sandman.rst |- requirements.txt |- your$_{p}ackage||--_{i}nit_{.py||--your_{p}ackage.py||--test||--models.py||--test_{y}our_{p}ackage.py|-setup.pyStep2/4:Gi$
2. Create a .pypirc configuration file in $HOMEdirectory$
[distutils] index-servers = pypi
[pypi] repository=https://pypi.python.org/pypi username=your$_{u}sernamepassword =$ $your_{p}assword3.ChangeyourMANIFEST.in$
recursive-include project$_{f}older * 4.UploadyourpackagetoPyPI$
python setup.py register -r pypi python setup.py sdist upload -r pypi Step 4/4:
Conda 2 1. Install conda tools
conda install conda-build conda install anaconda-client 2. Build a simple package
with conda skeleton pypi
cd your$_{p}ackage_{f}oldermkdircondacdcondacondaskeletonpypiyour_{p}ackageThiscreatesadirectorynamedyour_{p}$
|- your$_{p}ackage|| - -bld.bat|| - -meta.yaml|| - -build.sh3.Buildyourpackage$
conda build your$_{p}ackage$
convert to all platform conda convert -f –platform all C:-bld-64$_{p}ackage-0.1.1-py27_{0}.tar.bz24.UploadpackagestoAnaconda$
anaconda login anaconda upload linux-32/your$_{p}ackage.tar.bz2anacondauploadlinux-64/your_{p}ackage.tar.bz2anacondauploadwin-32/your_{p}ackage.tar.bz2anacondauploadwin-64/your_{p}ackage.tar.bz2CreatePrivateModuleStep1 : Makeyourmodule11.1pipinstallcookiecutter$
1.2 cookiecutter https://github.com/audreyr/cookiecutter-pypackage.git
1.3 Fill all necessary information
full$_{n}ame[AudreyRoyGreenfeld]$: $email[aroy@alum.mit.edu]$: $github_{u}sername[audreyr]$: $project_{n}ame[PythonBoilerplate]$: $project_{s}lug[]$: $project_{s}hort_{d}escription$: $release_{d}ate[]$: $pypi_{u}sername[]$: $year[2016]$: $version[0.1.0]$: $use_{p}ypi_{d}eployment_{w}ith_{t}ravis[y]$: $Step2 : BuildyourmoduleChangeyourMANIFEST.in$
recursive-include project$_{f}older * Buildyourmodulewithsetup.py$
cd your$_{p}roject_{f}older$

build local python setup.py build > It will create a new folder in > $PYTHON_HOME/Lib/sites-packages/your_project_name-0.1.0-py2.7.egg$
build distribution python setup.py sdist > It will create a zip file in $PROJECT_FOLDER/dist$ Step3 : $Usage your module In the same machine$
import $your_project_name In other machine$
Python: Build Install Local Package with Conda Here is a step by step tutorial about building a local module package install it from a custom channel 1 Step 1: Make a setup folder for your package with cookkiecutter on terminal:
mkdir build cd build pip install cookiecutter cookiecutter https://github.com/audreyr/cookiecutter-pypackage.git
Fill all necessary information
$full_name[AudreyRoyGreenfeld] : email[aroy@alum.mit.edu] : github_username[audreyr] : project_name[PythonBoilerplate] : project_slug[] : project_short_description : release_date[] : pypi_username[] : year[2016] : version[0.1.0] : use_pypi_deployment_with_travis[y] : It will create a directory$
|- LICENSE |- README.md |- TODO.md |- docs | |– conf.py | |– generated | |– index.rst | |– installation.rst | |– modules.rst | |– quickstart.rst | |– sandman.rst |- requirements.txt |- $your_package$ $||--_{init.py||--your_package.py||--test||--models.py||--test_your_package.py|-setup.pyCopyyourn$
Add this line to MANIFEST.in
recursive-include $project_folder * Step2 : Build conda package mkdir conda cd conda mkdir channel git clone https$ $//github.com/hunguyen1702/conda Build Local Template.git mv conda Build Local Template your_package_nam$ $rf.git README.md Edit the file meta.yaml with the instruction inside it cd .. conda build your_package_name Step$ $Create custom channel and install from local package Create a channel directory$
cd channel Convert $your_package you'vebuilt to all platform$
conda convert –platform all /anaconda/conda-bld/linux-64/$your_package_0$.1.0-$py27_0.tar.bz2$ and this will create :
channel/ linux-64/ package-1.0-0.tar.bz2 linux-32/ package-1.0-0.tar.bz2 osx-64/ package-1.0-0.tar.bz2 win-64/ package-1.0-0.tar.bz2 win-32/ package-1.0-0.tar.bz2 Register your package to your new channel
cd .. conda index channel/linux-64 channel/osx-64 channel/win-64 Veriy your new channel
conda search -c file://path/to/channel/ –override-channels If you see $your_package' s appearance, so it' s worked$
After that if you want to install that package from local, run this command:
conda install –use-local $your_package$
and when you want to create environment with local package from file, you just have export environment to .yml file and add this channels section before the dependencies section:
channels: - file://path/to/your/channel/

1.27 Production

Production with docker Base Image: magizbox/conda2.7/
Docker Folder
$your_app/appconfigmain.pyDockerfilerun.shDockerfile$
FROM magizbox/conda2.7:4.0
ADD ./app /app ADD ./run.sh /run.sh
RUN conda env create -f environment.yml run.sh
source activate $your_environment$
cd /app

python main.py Compose
service: build: ./service-app command: 'bash run.sh' Note: an other python conda with lower version (such as 3.5), will occur error when install requests package
python 3.4 hay 3.5
Có lẽ 3.5 là lựa chọn tốt hơn (phải có của tensorflow, pytorch, hỗ trợ mock)
Quản lý môi trường phát triển với conda
Chạy lệnh 'remove' để xóa một môi trường

```
conda remove --name flowers --all
```

1.28 Test với python

Sử dụng những loại test nào?
Hiện tại mình đang viết unittest với default class của python là Unittest. Thực ra toàn sử dụng 'assertEqual' là chính!
Ngoài ra mình cũng đang sử dụng tox để chạy test trên nhiều phiên bản python (python 2.7, 3.5). Điều hay của tox là mình có thể thiết kế toàn bộ cài đặt project và các dependencies package trong file 'tox.ini'
Chạy test trên nhiều phiên bản python với tox
Pycharm hỗ trợ debug tox (quá tuyệt!), chỉ với thao tác đơn giản là nhấn chuột phải vào file tox.ini của project.

1.29 Xây dựng docs với readthedocs và sphinx

20/12/2017: Tự nhiên hôm nay tất cả các class có khai báo kế thừa ở project languageflow không thể index được. Vãi thật. Làm thằng đệ không biết đâu mà build model.
Thử build lại chục lần, thay đổi file conf.py và package_reference.rst chán chê không được. Giả thiết đầu tiên là do hai nguyên nhân (1) docstring ghi sai, (2) nội dung trong package_reference.rst bị sai. Sửa chán chê cũng vẫn thế, thử checkout các commit của git. Không hoạt động!
Mất khoảng vài tiếng mới để ý thằng readthedocs có phần log cho từng build một. Lần mò vào build gần nhất và build (mình nhớ là) thành công cách đây 2 ngày
Log build gần nhất

```
Running Sphinx v1.6.5
making output directory...
loading translations [en]... done
loading intersphinx inventory from https://docs.python.org/objects.inv
    ↪ ...
intersphinx inventory has moved: https://docs.python.org/objects.inv ->
    ↪ https://docs.python.org/2/objects.inv
loading intersphinx inventory from http://docs.scipy.org/doc/numpy/
    ↪ objects.inv...
intersphinx inventory has moved: http://docs.scipy.org/doc/numpy/
    ↪ objects.inv -> https://docs.scipy.org/doc/numpy/objects.inv
building [mo]: targets for 0 po files that are out of date
```

```
building [readthedocsdirhtml]: targets for 8 source files that are out
    ↪ of date
updating environment: 8 added, 0 changed, 0 removed
reading sources ... [ 12%] authors
reading sources ... [ 25%] contributing
reading sources ... [ 37%] history
reading sources ... [ 50%] index
reading sources ... [ 62%] installation
reading sources ... [ 75%] package_reference
reading sources ... [ 87%] readme
reading sources ... [100%] usage

looking for now-outdated files... none found
pickling environment... done
checking consistency ... done
preparing documents... done
writing output... [ 12%] authors
writing output... [ 25%] contributing
writing output... [ 37%] history
writing output... [ 50%] index
writing output... [ 62%] installation
writing output... [ 75%] package_reference
writing output... [ 87%] readme
writing output... [100%] usage
```

Log build hồi trước

```
Running Sphinx v1.5.6
making output directory...
loading translations [en]... done
loading intersphinx inventory from https://docs.python.org/objects.inv
    ↪ ...
intersphinx inventory has moved: https://docs.python.org/objects.inv ->
    ↪ https://docs.python.org/2/objects.inv
loading intersphinx inventory from http://docs.scipy.org/doc/numpy/
    ↪ objects.inv...
intersphinx inventory has moved: http://docs.scipy.org/doc/numpy/
    ↪ objects.inv -> https://docs.scipy.org/doc/numpy/objects.inv
building [mo]: targets for 0 po files that are out of date
building [readthedocs]: targets for 8 source files that are out of date
updating environment: 8 added, 0 changed, 0 removed
reading sources ... [ 12%] authors
reading sources ... [ 25%] contributing
reading sources ... [ 37%] history
reading sources ... [ 50%] index
reading sources ... [ 62%] installation
reading sources ... [ 75%] package_reference
reading sources ... [ 87%] readme
reading sources ... [100%] usage

/home/docs/checkouts/readthedocs.org/user_builds/languageflow/
```

```
        ↪ checkouts/develop/languageflow/transformer/count.py:docstring
        ↪ of languageflow.transformer.count.CountVectorizer:106:
        ↪ WARNING: Definition list ends without a blank line; unexpected
        ↪ unindent.
/home/docs/checkouts/readthedocs.org/user_builds/languageflow/
        ↪ checkouts/develop/languageflow/transformer/tfidf.py:docstring of
        ↪ languageflow.transformer.tfidf.TfidfVectorizer:113: WARNING:
        ↪ Definition list ends without a blank line; unexpected unindent.
../README.rst:7: WARNING: nonlocal image URI found: https://img.
        ↪ shields.io/badge/latest-1.1.6-brightgreen.svg
looking for now-outdated files... none found
pickling environment... done
checking consistency ... done
preparing documents... done
writing output... [ 12%] authors
writing output... [ 25%] contributing
writing output... [ 37%] history
writing output... [ 50%] index
writing output... [ 62%] installation
writing output... [ 75%] package_reference
writing output... [ 87%] readme
writing output... [100%] usage
```

Đập vào mắt là sự khác biệt giữa documentation type
Lỗi

```
building [readthedocsdirhtml]: targets for 8 source files that are out
    ↪ of date
```

Chạy

```
building [readthedocs]: targets for 8 source files that are out of date
```

Hí ha hí hửng. Chắc trong cơn bất loạn sửa lại settings đây mà. Sửa lại nó trong phần Settings (Admin gt; Settings gt; Documentation type)

Khi chạy nó đã cho ra log đúng

```
building [readthedocsdirhtml]: targets for 8 source files that are out
    ↪ of date
```

Nhưng vẫn lỗi. Vãi!!! Sau khoảng 20 phút tiếp tục bấn loạn, chửi bới readthedocs các kiểu. Thì để ý dòng này
Lỗi

```
Running Sphinx v1.6.5
```

Chạy

```
Running Sphinx v1.5.6
```

Ngay dòng đầu tiên mà không để ý, ngu thật. Aha, Hóa ra là thằng readthedocs nó tự động update phiên bản sphinx lên 1.6.5. Mình là mình chúa ghét thay đổi phiên bản (code đã mệt rồi, lại còn phải tương thích với nhiều phiên bản nữa thì ăn c** à). Đầu tiên search với Pycharm thấy dòng này trong 'conf.py'

> *# If your documentation needs a minimal Sphinx version, state it here.*
> *# needs_ sphinx = '1.0'*

Đổi thành

> *# If your documentation needs a minimal Sphinx version, state it here.*
> needs_sphinx = '1.5.6'

Vẫn vậy (holy sh*t). Thử sâu một tẹo (thực sự là rất nhiều tẹo). Thấy cái này trong trang Settings

Ờ há. Thằng đần này cho phép trỏ đường dẫn tới một file trong project để cấu hình dependency. Haha. Tạo thêm một file 'requirements' trong thư mục 'docs' với nội dung

> sphinx==1.5.6

Sau đó cấu hình nó trên giao diện web của readthedocs

Build thử. Build thử thôi. Cảm giác đúng lắm rồi đấy. Và... nó chạy. Ahihi

Kinh nghiệm
* Khi không biết làm gì, hãy làm 3 việc. Đọc LOG. Phân tích LOG. Và cố gắng để LOG thay đổi theo ý mình.
PS: Trong quá trình này, cũng không thèm build thằng PDF với Epub nữa. Tiết kiệm được bao nhiêu thời gian.

Phần II

Linh tinh

Chương 2

Trở thành giảng viên

Hơi bị hay Ai cũng có thể trở thành giảng viên giỏi nếu của anh Cường tại techmaster. Tưởng phải mua, hóa ra lại được set free. A hihi.

- Lên ý tưởng
- Chuẩn bị nội dung
- Code mẫu
- Xây dựng kịch bản
- Trình bày đơn giản, dễ hiểu

Một bài giảng

2.1 Khác biệt giữa dạy online và offline

Dạy online
Ưu điểm

1. Khả năng mở rộng rất tốt
2. Chi phí thấp
3. Số lượng học viên không giới hạn

Nhược điểm

- Tỷ lệ bỏ học rất cao
- Tương tác chưa thực sự tốt

2.1.1 Làm sao để giảm tỷ lệ bỏ học trực tuyến?

- Video ngắn $<$ 10 phút
- Cấp chứng chỉ 70-90 USD lấy 1 chứng chỉ cho khóa học 4-6 tuần
- Chấm bài tập

- Facebook group kết nối giảng viên học viên
- Bài tập đủ dễ: chia nhỏ dự án khó ra
- Gamification: học như chơi
- Chịu khó email thông báo cho lớp

2.2 Marketing

Các hình thức marketing
1. Mạng xã hội: Facebook, forum Phải boost quảng cáo. 500000 - 2000000 VND cho mỗi khóa học
2. SEO - Google Search
3. Qua hội thảo, meetup hiệu quả thấp do số lượng người tham gia < 40 người
Conversation rate: số người mua hàng / số người tham quan
Nếu đến từ organic search CR = 1 / 200
Nếu qua chia sẻ bài viết mạng xã hội CR = 1 / 4000
Nếu qua giới thiệu, tư vấn người quen CR = 1 / 2

2.2.1 Chia sẻ bài học trên Facebook

Hoàn thành bài học
Một bài viết chia sẻ lên Facebook có ảnh sẽ được người dùng click vào cao gấp 2-3 lần bài viết không có ảnh. Một bài viết có ảnh thumbnail có nút play xem video thì sẽ có số người dùng click vào cao hơn 1.3 lần so với ảnh thuần túy
Một khóa học online không giới thiệu qua Facebook, YouTube thì chắc chắn không có học viên dù giảng viên giỏi đến mấy
Bài giới thiệu trên facebook có nút play có lượt click lớn hơn nhiều so với không có nút play

2.3 Flip Learning

Lớp học đảo ngược
Vấn đề học online

- Bỏ học cao
- Tương tác face 2 face kém
- Thực hành không có, không kiểm soát

Vấn đề học offline

- Chi phí cao
- Giao thông không thuận tiện
- Tuyển sinh khó

Flip Learning lớp học đảo ngược, kết hợp ưu điểm học trực tuyến và thực hành phòng lab.
- Khuyến khích học viên xem trước bài giảng video hướng dẫn giảng viên
- Đọc tìm hiểu thêm, trả lời trắc nghiệm qua Internet
- Tại buổi học, dành tối đa thời gian để **thực hành, hỏi đáp, hợp tác**
Khi đến lớp, học viên đã có kiến thức, biết rõ mình sẽ làm gì.
- Không thụ động, rèn tính tự học, tự đọc
- Không chống cằm nhìn giảng viên, không ghi chép
- Hỏi, làm, nói, giúp đỡ nhau
Flip Learning:
- Giảng viên phải chuẩn bị bài giảng
- Đến lớp trao đổi, hướng dẫn

2.4 Thuyết trình

Tự nhiên hôm nay (22/01/2018) lại đọc được bài You suck at PowerPoint, thấy hay quá.
Sau đây là 10 lỗi thường gặp khi làm bài thuyết trình

1. Quá nhiều chữ trong 1 slide.
2. Màu chữ và màu nền không tương phản với nhau.
3. Dùng clip art, word art.
4. Hình ảnh sử dụng trong slide chất lượng kém, scale sai tỉ lệ.
5. Sử dụng nhiều font chữ trong 1 slide.
6. Lạm dụng quá nhiều hiệu ứng (animation/transition).
7. Bài presentation không có cấu trúc.
8. Slide không ăn nhập gì với nội dung trình bày.
9. Không ghi rõ nguồn khi sử dụng tài liệu, hình ảnh của người khác.
10. Ý thức của người làm slide

2.5 Xác định đối tượng học tập

Học viên Techmaster là ai?
1- Sinh viên CNTT năm đầu, cần thực hành: 302- Người thất nghiệp: 403- Lập trình cần nâng cao kỹ năng 304- Hầu hết là từ nông thôn 705- Thời gian có hạn, cần tìm việc ngay
Họ cần gì?
1- Bài giảng thực tế, có nhiều ví dụ sinh động 2- Rất khác với ở trường đại học: + Không phải thi + Lý thuyết ít, thực hành nhiều, trừu tượng ít + Ví dụ phải cool + Có người hỗ trợ ngay 3- Trung tâm giới thiệu việc làm sau khi học

2.6 Sai lầm cỗ hữu của giảng viên

1- Nghĩ ai cũng biết như mình.
Học viên 80Giảng viên dùng toàn từ chuyên ngành tiếng Anh mà không giải thích cặn kẽ
2- Nói nhiều, lý thuyết nhiều, ít có ví dụ, thí nghiệm minh họa. Học viên đã rất chán kiểu dạy ở đại học
3- Khô cứng: giải thích về kế thừa
4- Ba hoa, bốc phét như bán hàng đa cấp. Lesson online giới hạn 10 phút, học offline chỉ có 140 phút.
5- Dùng nhiều tính từ, đại ngôn: rất, cực.. mà không có con số.
6- Cẩu thả: không chuẩn bị kỹ slide, slide toàn chữ 12 trang.
7- Mã nguồn, dự án ví dụ vô duyên, khó hiểu

```
*
**
***
****
*****

*     *
 *   *
  * *
   *
```

8- Chia nhỏ các bài tập

2.7 Viết kịch bản nói

1- Viết chi tiết chính xác đến từng câu, rồi đọc lại + Nói chưa lưu loát > viết chi tiết 2- Liệt kê ý chính, vừa demo, vừa nói + Nói tốt + lười + Hậu kỳ phải cắt bỏ nhiều đoạn nói nhịu, ề à Câu từ dài dòng, rườm ra, tỷ lệ tiếp thu càng thấp Học trực tuyến, học viên dùng mắt đọc - xem là chính 70%, nghe là phụ, 30%. Video qua 5 phút gây buồn ngủ.
Nhắc kịch bản
- Không nên in ra giấy ! - Màn hình rộng Dell Utra, để mở cửa sổ nhắc kịch bản - Mở rộng thêm 1 màn hình mới - Gõ kịch bản ra iPad, dựng iPad lên thành màn hình phụ

2.8 Đồ nghề tạo giáo trình trực tuyến

1- Web Cam
+ Logitech C920 hoặc Xioami Camera loại 650,000VND. + Tỷ lệ bỏ lửng bài video giảm 30% nếu học viên thấy mặt giảng viên. + Mặt giảng viên đẹp trai, hấp dẫn, hài hước càng tốt + Mở sẵn cửa sổ ứng dụng cần demo. Giảm tối đa thời gian chết
2- Camtasia
+ Camtasia Studio trên Windows có nhiều chức năng hơn bản Camtasia Mac
+ Phiên bản Camtasia Mac không nén được video định dạng H264 phải dùng

FFmpeg để nén + Thu ở khung hình 1280 * 720 pixels. Trên Mac có công cụ xScope để căn kích thước màn hình thu
3- Phòng thu âm cần yên tĩnh
+ Đóng chặt cửa sổ, cửa phòng khi quay chấp nhật nóng trong 10 phút quay.
+ Bật chế noise remove trong Camtasia
4- Sublime Text 5- Ink2Go 6- Lucid Chart 7- Adobe Photoshop 8- Skitch 9- PowerPoint 10- Slides.com

2.9 Kinh nghiệm khi quay video

Phòng cần yên tĩnh: nên tắt quạt hay các thiết bị điện tử gây ồn. Tắt điện thoại tránh cuộc gọi đến khi đang ghi hình Để micro xa bàn phím để không lẫn tiếng lạch cạch
Tuyệt đối không nên để background wallpaper lòe loẹt, học viên không chú ý vào demo của bạn mà xem background thì hiệu quả truyền đạt rất tệ. Tốt nhất để màn hình background là đen 100
Tắt Skype, chat, loại bỏ các icon không cần thiết trên màn hình desktop
Nên dùng web camera Logitech C920 để xa mồm để không ghi tiếng thở hổn hển của giảng viên
Nên quay cả mặt giảng viên, nhưng cut crop để không quay background của phòng thu. Hãy để học viên tập trung vào bài giảng
Không sử dụng âm thanh nên có tiếng hát hoặc giai điệu nhanh, phức tạp khiến học viên không tập trung
Lưu ý rằng:
Học viên thích xem demo sinh động và đọc chữ trên màn hình rất nhanh, đừng nói quá dài dòng, phức tạp.

2.10 Thiết kế để học trực tuyến

1- Một video $<$ 10 phút. Tốt nhất 3 - 5 phút
2- Ví dụ là một dự án mẫu chạy được. Đơn giản tối đa
3- Học viên phấn khích khi thấy kết quả cuối cùng
4- Chia nhỏ task top-down:
+ Sản phẩm này làm gì?
+ Có chức năng gì? chỉ nên 1 hoặc 2
+ Màn hình chính

2.11 Upload video và tạo bài giảng

Tạo một lesson như thế nào?
- Có một video dài $<$ 10 phút
- Tối thiểu 3 câu hỏi quiz
- Nên bổ sung mã nguồn hoặc slide

2.12 Tính logic cấu trúc bài giảng

Không nên:

- Tùy tiện theo ý thích hoặc kinh nghiệm cá nhân giảng viên - Giập khuôn theo sách hoặc khóa học có sẵn Hậu quả học nhiều, nhưng kỹ năng, kiến thức lộn xộn. Thà học ít mà dùng được nhiều, hiệu quả còn hơn
Nên:
- Thiết kế top down dùng Mindmap - Các mẩu kiến thức có liên kết, sâu chuỗi
- Trước - Sau: thủ tục -> hướng đối tượng -> design pattern array -> dictionary -> linked list
SQL raw query -> Objection Relation Mapping
- Kế thừa: Cha - Con, Chị - Em
UIView > UIScollView > UITableView, UICollectionView
- Đối lập - so sánh:
Postgresql <> MongoDB, FireBase <> RethinkDB Apple Push Notification <> Google Cloud Messaging
- Phân nhóm theo tiêu chí: Dễ học, dùng thường xuyên, học viên sướng. Sướng quyết định tất cả
Ví du:
- Lẻ tẻ, minh họa cụ thể từng cú pháp, API function. Học viên không thu hoạch được nhiều - Một dự án kết hợp các bước sẽ thú vị hơn, dài hơn. Hay lúc đầu, buồn ngủ lúc sau.

2.13 Đặt câu hỏi cho học viên

Nếu không đặt câu hỏi:
- Học viên thụ động, buồn ngủ, sớm rời bỏ khóa học - Giảng viên không biết học viên có hiểu bài hay không?
Đặt câu hỏi như thế nào?
1- Tránh Yes / No Question 2- Câu hỏi để thức tỉnh khả năng suy nghĩ, động não
AJAX được ai phát minh ra?
AJAX là viết tắt của cụm từ nào?
Một trang web hiển thị giá cổ phiếu cập nhật 15 giây một lần. Có 4000 khách hàng cùng truy cập, vậy có bao nhiêu AJAX gửi về máy chủ trong 1 giây?
3- Câu hỏi ôn lại kiến thức
Ý nghĩa của http verb GET, POST, PUT, DELETE là gì, khác nhau như thế nào? Câu hỏi rất rộng. Nhưng khi nào dùng POST và khi nào dùng PUT sẽ tập trung hơn và dễ tạo lựa chọn để trả lời. Sử dụng GET truyền id có thể thay thế DELETE được không? Tại sao không nên.
4- Hãy nhìn vào mắt học viên khi hỏi. (offline)
Hỏi những học viên có dấu hiệu buồn ngủ

2.14 Slide ngắn gọn, xúc tích

Video < 10 phút

2.15 Nén video chuẩn H.264

Áp dụng đối với hệ điều hành Mac và Linux. Sau khi ghi hình ở độ phân giải 1280x720 pixels, và xuất ra video mp4. Nếu kích thước video tính theo

Megabytes > 2 * số phút (Mb) thì video đó nén chưa tốt. Chúng ta cần phải nén video với chuẩn H264.
Các bước thực hiện
Vào web site này https://www.ffmpeg.org/ tải phần mềm ffmpeg về
Copy vào thư mục /usr/local/bin để có thể gọi ffmpeg ở mọi nơi Tạo một bash script có tên v264 trong thư mục /usr/local/bin cd /usr/local/bin nano v264
Paste đoạn script này vào file v264

```
#!/bin/bash
if [ -z "$1" ]; then
    echo "You must input video file name"
    exit
fi
if [ -f $1 ]; then
        output="$(echo $1 | sed 's/\.[^.]*$//')"
        ffmpeg -i $1 -c:a copy -x264-params crf=30 -b:a 64k "
            ↪ $output-264.mp4"
else
        echo "File $1 does not exist"
fi
```

Để file v264 thực thi được cần gõ lệnh để bổ xung quyền execution

```
chmod +x v264
```

Khi nén video gõ lệnh này

```
v264 yourvideo.mp4
```

2.16 Các lỗi giảng viên thường mắc phải khi làm video

1. Nói nhiều, thích ôm đồm nhiều thông tin trong một video > Độ dài tuyệt nhất là 3 phút
2. Chữ bé, các chi tiết khó nhìn khiến người dùng bị mỏi mắt > Thường sau 3-4 phút là khá mệt
3. Thiếu dòng title phút đầu tiên của video clip -> người dùng khó định hình ngày giảng viên muốn nói gì
4. Không điểm danh các mục nhỏ, nội dung trình bày trong video > Không biết sẽ đi về đâu
5. Không có điểm nhấn, phân cách giữa các mục nhỏ trong hướng dẫn > Mình đang ở mục mấy rồi nhỉ? Còn mấy mục nữa thì hết
6. Ít hình minh họa > Một hình ảnh dùng đúng hiệu quả hơn 1000 câu giải thích
7. Chỉ gõ lệnh nhưng ít khi nói mục đích để làm gì? Kết quả mong muốn là gì
8. Không giải thích các thuật ngữ mới
9. Chưa tính đến môi trường thực hành của học viên có thể khác so với giảng viên

2.17 Sử dụng Photoshop

Sử dụng Photoshop để làm poster cho khóa học

2.18 Chú ý khi quay video

1- Camstasia 3 đã ra mắt
2- Ink2 Go
3- xScap
4- Terminal: font Roboto Mono for Powerline 18pt
5- Powerpoint 720p

2.19 Thất bại khi thực hành

Mọi lỗi sơ suất đều khiến giảng viên cực kỳ bối rối, áp lực Mồ hôi toát ra như tắm, học viên xì xào, chán nản vì sốt ruột. Buổi học thất bại
Mọi demo cần chuẩn bị tập dượt, dù là chi tiết nhỏ nhất.
Thực hành tối thiểu 3 lần, ghi lại các bước thành Hand-On-Lab. Ghi lại những lỗi giảng viên đã gặp phải:
- Cắm nhầm cực dương - âm tụ hóa
- Thiếu tụ lọc nhiễu
- Máy học viên có thể bị thiếu RAM, cấu hình sai
Giảng viên phát trước Hand-On-Lab cho học viên xem. Tạo câu hỏi quiz để đảm bảo học viên hiểu bài trước khi thực hành.
Đối với những bài thực hành khó, hãy yêu cầu học viên cài sẵn môi trường thực hành bởi:
1- Thời gian tải phần mềm ở lớp học có thể lâu 2- Tự học viên phát hiện lỗi khi cài phần mềm ở nhà 3- Học viên được làm quen trước với bài lab đơn giản
Đối với bài thực hành khó, hãy mạnh dạn chọn ra 1-2 học viên giỏi để chuẩn bị trước. Đây là đặc ân mà học viên giỏi sẽ thích thú và cố gắng tham gia.

2.20 Học viên mất căn bản - Trả lời hỏi đáp

Question: Trong một lớp học có quá nhiều học viên không có căn bản để học, nhưng lại có vài học viên giỏi, tiếp thu rất nhanh.
Phục vụ học viên mất căn bản hay phục vụ học viên giỏi?
Trả lời: Đừng chia thành 2 nhóm học viên giỏi và học viên kém. Nhóm giỏi sẽ tiếp tục đi nhanh hơn và nhóm kém sẽ bỏ cuộc.
Chia nhóm để ghép học viên giỏi và kém. Học viên giỏi sẽ giúp đỡ học viên kém.
Lớp 12 học viên chia thành 4 nhóm. Tạo ra cuộc thi đua nhỏ giữa các nhóm.
Nếu có học viên bỏ học, sắp xếp lại nhóm.
Question: Trong một lớp học thực hành, nếu có nhiều câu hỏi ở mức độ khó khác nhau? Giảng viên chỉ được chọn trả lời một. Vậy chọn câu trả lời nào?
Answer: Hãy trả lời có khả năng hài lòng số đông học viên nhất.
Câu hỏi khó trả lời buổi sau, qua Facebook group của lớp
Câu hỏi ngoài lề không trả lời
Câu hỏi dễ, nhờ học viên khá trong lớp trả lời hộ

2.21 Bước 1: Chuẩn bị giáo trình

Mục tiêu của bước này là xây dựng khung giáo trình cho toàn bộ khóa học.
Các điều cần ghi nhớ

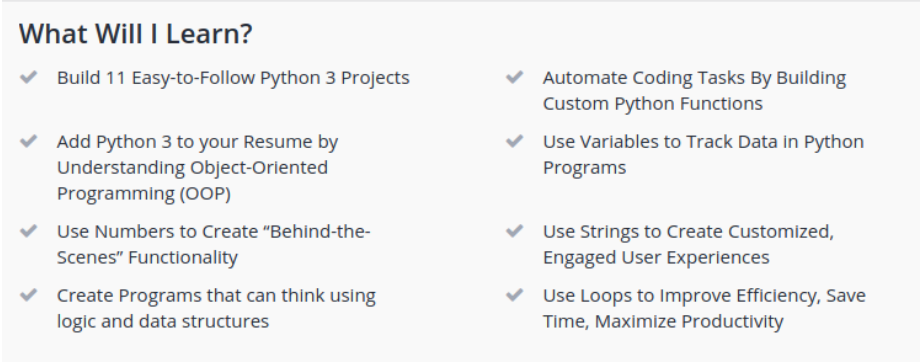
- Top Down tốt hơn Bottom Up

2.21.1 Sử dụng MindMap

Một công cụ hữu ích là MindMup, có thể dễ dàng tích hợp trong Google Drive

2.21.2 Mục tiêu của khóa học

Hãy xem một mục tiêu cực kỳ hấp dẫn của một khóa học về Python rất thành công trên udemy.com



Có lẽ điểm cộng quan trọng nhất trong khóa học này là **Xây dựng 11 dự án Python 3**

2.21.3 Tham khảo sách ebook, Udemy, PluralSights, Lynda...

3- Người bình thường không thể nhớ quá 5 mục trong một buổi học: + Offline: 1 buổi dạy 1 chủ đề chia thành 4 minitask + Online: 1 section gồm 4-5 video, mỗi video $<$ 10 phút
4- Sử dụng MindMap

Tài liệu tham khảo

- Xây dựng khung giáo trình, Trịnh Minh Cường

Tài liệu tham khảo

Ghi chú