Pydanny Presentations Documentation

Release 44

Daniel Greenfeld

Sep 27, 2017

Contents

Contents:

CHAPTER 1

Intro to Python

Info Your basic hour long introduction to the Python programming language.

Author Daniel Greenfeld (http://github.com/pydanny)

Upcoming

Scale 10x

Abstract

Python is often considered a "new" scripting language but has been around for over 20 years. This talk will go over who uses Python, how Python is similar and different to other languages, core concepts of Python, basic language constructs, live code examples, web framework overview, the Python Package Index, the Southern California Python community, and will finish up with some of the more exciting and interesting contributions and efforts of the Python community. Technical content will include: REPL, Types, Mutable/Immutable, Getting help, Lists, Dictionaries, Functions, Conditionals & booleans, Whitespace, Iteration, Slicing, I/O, Classes, Exceptions, Packaging and layout.

Contents:

Slides TOC

- Opener
  - Subtitle: 20 cool things you can do with Python
- Who am I
  - History
  - Credentials

- Who uses Python?
  - Everyone!
  - Who doesn't use Python?
  - NASA, Canonical in Ubuntu, Google, et al
  - Startups, Scientists, Animation Studios, et al
- What is Python?
  - 20 years old and mature
  - A programming language named after Monty Python
  - Dynamically typed scripting language
- Python is similar to:
  - Ruby
  - Lisp
  - et al
- Python is different than:
  - Ruby
  - Java
  - Lisp
- Core concepts
  - Philosophy of Core devs
    - Conservative growth
    - *We read Knuth so you don't have to*
    - Aim for simple implementation
  - Zen of Python
    - A few of my favorite lines
  - PEP-8
- Which Python to use?
- 20 Cool things you can do with Python

1. Learn it fast
   - Experienced developers take a week to get competent in Python
   - Java takes 6 months
   - C takes 2 years
2. Introspect
   - String basic type
   - dir
   - help
3. Play with Strings

    - Concatenation
    - String formatting
    - Join iterables
4. Express yourself to laymen
    - Don't know Python? Bet you can read this!
5. Play with the REPL
    - Standard python shell
    - ipython
    - bpython
6. List Comprehension
    - [x for x in range(15)]
    - List Generators for ultimate power
7. Generate Exceptions
8. Create small, isolated environments for installing packages
        - Good for testing different versions of the same library:

    easy_install pip pip install virtualenv virtualenv my_env source my_env/bin/activate
9. Generate the code coloration used in these slides.
    - http://pygments.org/
    - Show demo
    - Mention that both bitbucket and github use Pygments to colorize code.
10. Go faster than C and Javascript
    - Everyone knows C is fast, right?
    - And JavaScript on V8 is really fast too.
    - PyPy, an implementation of Python written in Python, runs faster.
    - Faster than JavaScript, in some cases faster than C.
    - Benchmarks, et al
    - http://en.wikipedia.org/wiki/PyPy
    - http://speed.pypy.org/
    - http://morepypy.blogspot.com/2012/01/numpypy-progress-report-running.html
    - http://blog.bossylobster.com/2011/08/lesson-v8-can-teach-python-and-other.html
    - Quora is the best known prominent user at this time
    - Integration with numpy and other scientific libraries in progress
11. Persist
    - LRU caching
    - Connect to SQL databases
    - Connect to NoSQL

12. Build websites
13. Internationalize
    - unicode
    - gettext implementation

Code samples

Executed and displayed here so I can cut-and-paste into Keynote.

Whitespace

```
""" whitespace.py """
from random import randrange

def numberizer():
    # Generate a random number from 1 to 10.
    return randrange(1, 11)

number = numberizer()
if number > 5:
    print("This number is big!")

class RandomNumberHolder(object):
    # Create and hold 20 random numbers using numberizer

    def __init__(self):
        self.numbers = [numberizer(x) for x in range(20)]

random_numbers = RandomNumberHolder()
```

Zen of Python

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
```

```
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Strings

```
>>> foo = 'bar' # TODO: strike-out in slides
>>> spam = 'eggs'
```

String methods

```
>>> fun = 'spam and EGGS   '
>>> dir(fun)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
 '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
 '__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt__',
 '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
 '__subclasshook__', '_formatter_field_name_split', '_formatter_parser',
 'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs',
 'find', 'format', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace',
 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace',
 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
 'zfill']
>>> fun.strip()
'spam and EGGS'
>>> spam.title()
'Spam And Eggs   '
>>> fun.capitalize()
'Spam and eggs   '
>>> fun.index('3')
3
>>> len(fun)
16
>>> fun.__len__() # same as the len function
16
>>> help(fun)
no Python documentation found for 'spam and EGGS   '
>>> help(str)
```

Help Slide Part I

code:

```
Help on class str in module __builtin__:

class str(basestring)
 |  str(object) -> string
 |
 |  Return a nice string representation of the object.
 |  If the argument is a string, the return value is the same object.
 |
 |  Method resolution order:
```

```
|      str
|      basestring
|      object
|
|  Methods defined here:
|
|  __add__(...)
|      x.__add__(y) <==> x+y
|
|  __contains__(...)
|      x.__contains__(y) <==> y in x
```

Help Slide Part II

code:

```
|  capitalize(...)
|      S.capitalize() -> string
|
|      Return a copy of the string S with only its first character
|      capitalized.
|
|  center(...)
|      S.center(width[, fillchar]) -> string
|
|      Return S centered in a string of length width. Padding is
|      done using the specified fill character (default is a space)
|
|  count(...)
|      S.count(sub[, start[, end]]) -> int
|
|      Return the number of non-overlapping occurrences of substring sub in
|      string S[start:end].  Optional arguments start and end are interpreted
|      as in slice notation.
```

Play with strings

Strings are immutable

sourcecode:: python

```
>>> a = "Daniel"
>>> b = "Adam"
>>> c = "Greenfeld"
>>> a + b + c
'DanielAdamGreenfeld'
>>> "{0} {1} {2}".format(a, b, c)
'Daniel Adam Greenfeld'
>>> "{first} {middle} {last}".format(first=a, middle=b, last=c)
'Daniel Adam Greenfeld'
>>> lst = [a,b,c]
>>> lst
['Daniel', 'Adam', 'Greenfeld']
>>> " ".join(lst)
'I like Python'
```

Diving into lists

```
lst = [1,2,3,4,5,"Python", [1,2,3,4]]
```

Sets dominate

```
>>> lst = [1,1,1,1,1,2,2,2,3,3,3,3,3,3]
>>> s = set(lst)
>>> s
set([1,2,3])
```

How is this useful? What about counting unique words in a paragraph?

```
>>> address = """Four score and seven years ago our fathers brought forth on this␣
↪continent a new nation..."""
>>> for r in [',','.','-']:
...     address = address.replace(r,'')
>>> words = address.split(' ')
>>> len(words)
278
>>> unique_words = set(words)
>>> len(unique_words)
143
```

List Comprehension

```
>>> items = [x for x in range(20)]
>>> items
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> [x for x in range(20) if x % 2]
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
>>> # Fizzbuzz solved using Python's List Comprehension
>>> lst = [(x, 'Fizz', 'Buzz', 'FizzBuzz') \
...     [(not x % 3) | (not x % 5) << 1] for x in range(20)]
>>> for x in lst: print(x)
FizzBuzz
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
```

```
Fizz
19
```

Generator Expressions

```
>>> items = (x for x in range(20))
>>> items
<generator object <genexpr> at 0x100721460>
```

In Python, a generator can be thought of as an iterator that contains a frozen stack frame. Whenever the iterator's next() method is called, Python resumes the frozen frame, which executes normally until the next yield statement is reached. The generator's frame is then frozen again, and the yielded value is returned to the caller.

```
# loop way
wwwlog = open("access-log")
total = 0
for line in wwwlog:
    bytestr = line.rsplit(None,1)[1]
    if bytestr != '-':
        total += int(bytestr)
print "Total", total
```

```
# generator expressions way
wwwlog     = open("access-log")
bytecolumn = (line.rsplit(None,1)[1] for line in wwwlog)
bytes      = (int(x) for x in bytecolumn if x != '-')
print "Total", sum(bytes)
```

Pygments

First get the pygments code:

```
pip install pygments
```

Then run this simple 'recursive' program:

```
from pygments import highlight
from pygments.lexers import get_lexer_by_name
from pygments.formatters import HtmlFormatter

if __name__ == '__main__':
    code = open("pygments_demo.py", "rw").read()
    lexer = get_lexer_by_name("python", stripall=True)
    formatter = HtmlFormatter(linenos=False, cssclass="source")
    css = HtmlFormatter().get_style_defs('.source')
    highlighted_code = highlight(code, lexer, formatter)
    page = """
        <html>
            <head><style>{css}</style></head>
            <body>{highlighted_code}</body>
        </html>
        """.format(css=css, highlighted_code=highlighted_code)
    print(page)
```

Now run the program and check the results:

```
python pygments_demo.py > text.html
open text.html
```

Isolate Environments

code:

```
$ curl https://raw.github.com/pypa/pip/master/contrib/get-pip.py | python
$ pip install virtualenv
$ virtualenv my_env
$ source my_env/bin/activate
(my_env) $
```

More code:

```
(my_env) $ pip install django==1.3.1
(my_env) $ pip install requests==0.9.1
(my_env) $ pip install mongoengine==0.5.2
(my_env) $ pip install celery==2.4.6
(my_env) $ pip freeze
celery==2.4.6
django==1.3.1
mongoengine==0.5.2
requests==0.9.1
(my_env) $ pip freeze > requirements.txt
...
(another_env) $ pip install -r requirements.txt
```

Persist SQL

```
from datetime import datetime

from django.contrib.auth.models import User
from django.db import models
from django.utils.translation import ugettext_lazy as _

class Post(models.Model):

    author = models.ForeignKey(User)
    title = models.CharField(_('Title'), max_length=100)
    content = models.TextField(_("Content"))
    pub_date = models.DateTimeField(_("Publication date"))

class Comment(models.Model):
    post = models.ForeignKey(Post)
    name = models.CharField(_('Title'), max_length=100)
    content = models.TextField(_("Content"))
```

Persist NoSQL

```
from django.utils.translation import ugettext_lazy as _

import mongoengine as me

class User(me.Document):
    email = me.StringField(_('email'), required=True)
    first_name = me.StringField(_('first name'), max_length=30)
    last_name = me.StringField(_('last name'), max_length=30)

class Post(me.Document):
    title = me.StringField(_('title'), max_length=100, required=True)
    author = me.ReferenceField(User)
    content = me.StringField(_('content'))
    pub_date = me.DateTimeField(_("Publication date"))

class Comment(me.EmbeddedDocument):
    name = me.StringField(_('name'), max_length=100)
    content = me.StringField(_('content'))
```

Message Queues

dependencies:

```
celery
requests
```

tasks.py:

```
import logging
import requests
from celery import task

from products.models import Product

logger = logging.getLogger('products.tasks')

@task
def confirm_all_images():

    for product in Product.objects.all():
        response = request.get(product.medium_image.url)
        if response.status_code != 200:
            msg = "Product {0} missing image".format(product.id)
            logging.warning(msg)
```

Shell:

```
>>> from products.tasks import confirm_all_images
>>> result = confirm_all_images.delay()
>>> result.ready()
False
>>> result.ready()
True
```

Exception Handling

```
>>> import logging
>>> logger = logging.getlogger()
>>>
>>> class CustomTypeError(Exception):
...     pass

>>> try:
...     a = 1 + "Error"
>>> except TypeError as e:
...     raise CustomTypeError(e)
>>> except Exception as e:
...     logger.error(e)

Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
__main__.CustomTypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Generators

```
>>> def countdown(n):
...     print("Counting down from {0}".format(n))
...     while n > 0:
...         yield n
...         n -= 1

>>> x = countdown(10)
>>> x
<generator object at 0x58490>
>>> x.next()
Counting down from 10
10
>>> x.next()
9
>>> x.next()
8
>>> x.next()
7
```

- http://www.dabeaz.com/generators/Generators.pdf

Do things with Strings

```
>>> scale = 'Southern California Linux Expo'
>>> scale[0]
'S'
>>> scale[0:8]
'Southern'
>>> scale[:-11]
'Southern California Linux'
>>> scale[0:8] = 'Northern'
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

```
>>> scale.replace('Southern California','SoCal')
'SoCal Linux Expo'
>>> scale
'Southern California Linux Expo'
>>> s = scale.replace('Southern California','SoCal')
>>> s
'SoCal California Linux Expo'
>>> scale.startswith('Windows')
False
>>> scale.endswith('Windows')
False
>>> scale.startswith('Southern')
True
>>> 'Windows' in scale
False
>>> 'Linux' in scale
True
```

Basics

```
>>> x, y, z = 5, 10, 15
>>> 5 < 10
True
>>> 5 > 10
False
>>> True == False
False
>>> (5 == x) or (10 == x)
True
>>> (5 == x) and (10 == x)
False
>>> x + y - z
0
>>> 10 * 5
50
>>> 10 / 5
2
>>> 10 + 5
15
>>> 10 ** 2
100
```

Dictionaries

Python dictionaries are mutable key/value stores.

```
>>> data = {
        'name':'Daniel Greenfeld',
        'nickname':'pydanny',
        'states_lived':['CA','KS','MD','NJ','VA','AD'],
        'fiancee':'Audrey Roy'
        }
>>> data['name']
'Daniel Greenfeld'
>>> data['nickname'] = 'audreyr'
```

```
>>> data['nickname']
'audreyr'
>>> data['nickname'] = 'pydanny'
>>> data.keys()
['fiancee', 'nickname', 'name', 'states_lived']
>>> data.get('fiancee')
'Audrey Roy'
>>> data.get('fiance')
None
>>> data.pop('fiancee')
'Audreyr'
>>> data
{'nickname': 'pydanny', 'name': 'Daniel Greenfeld', 'states_lived': ['CA', 'KS', 'MD',
↪ 'NJ', 'VA']}
>>> data['fiancee'] = 'Audrey Roy'
>>> data
{'fiancee': 'Audrey Roy', 'nickname': 'pydanny', 'name': 'Daniel Greenfeld', 'states_
↪lived': ['CA', 'KS', 'MD', 'NJ', 'VA', 'AD']}
```

Work with JSON

```
>>> import json

>>> data = {
    'name':'Daniel Greenfeld',
    'nickname':'pydanny',
    'states_lived':['CA','KS','MD','NJ','VA','AD'],
    'fiancee':'Audrey Roy'
    }
>>> type(data)
<type 'dict'>
>>> payload = json.dumps(data)
>>> payload
'{"fiancee": "Audrey Roy", "nickname": "pydanny", "name": "Daniel Greenfeld", "states_
↪lived": ["CA", "KS", "MD", "NJ", "VA", "AD"]}'
>>> type(payload)
<type 'str'>
>>> restored = json.loads(payload)
>>> restored
<type 'dict'>
>>> restored
{u'fiancee': u'Audrey Roy', u'nickname': u'pydanny', u'name': u'Daniel Greenfeld', u
↪'states_lived': [u'CA', u'KS', u'MD', u'NJ', u'VA', u'AD'
]}
```

Object-Oriented Programming

```
class Animal(object):
    def __init__(self, name):     # Constructor of the class
        self.name = name
    def talk(self):               # Abstract method, defined by convention only
        raise NotImplementedError("Subclass must implement abstract method")

class Cat(Animal):
```

```
    def talk(self):
        return 'Meow!'

class Dog(Animal):
    def talk(self):
        return 'Woof! Woof!'

animals = [Cat('Missy'),
           Cat('Mr. Mistoffelees'),
           Dog('Lassie')]

for animal in animals:
    print animal.name + ': ' + animal.talk()
```

output:

```
Missy: Meow!
Mr. Mistoffelees: Meow!
Lassie: Woof! Woof!
```

- http://en.wikipedia.org/wiki/Polymorphism_in_object-oriented_programming#Examples

REPL

example:

```
$ python
Python 2.7.1+ (r271:86832, Apr 11 2011, 18:13:53)
[GCC 4.5.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

python:

```
>>> 3 + 4
7
>>> a = 5 * 10
>>> a
50
>>> def add(a, b):
...     return a + b
...
>>> add(3,4)
7
>>> add('Py','thon')
'Python'
>>> for x in 'Python':
...     print x
...
P
y
t
h
o
n
>>>
```

Serve the Web

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

PyMongo

```
>>> import pymongo
>>> connection = pymongo.Connection("localhost", 27017)
>>> db.name
u'test'
>>> db.my_collection
Collection(Database(Connection('localhost', 27017), u'test'), u'my_collection')
>>> db.my_collection.save({"x": 10})
ObjectId('4aba15ebe23f6b53b0000000')
>>> db.my_collection.save({"x": 8})
ObjectId('4aba160ee23f6b543e000000')
>>> db.my_collection.save({"x": 11})
ObjectId('4aba160ee23f6b543e000002')
>>> db.my_collection.find_one()
{u'x': 10, u'_id': ObjectId('4aba15ebe23f6b53b0000000')}
>>> for item in db.my_collection.find():
...     print item["x"]
...
10
8
11
>>> db.my_collection.create_index("x")
u'x_1'
>>> [item["x"] for item in db.my_collection.find().limit(2).skip(1)]
[8, 11]
```

Extra Code Samples

Basic list operations

You're going to enjoy lists.

```
>>> my_list = [1, 2, 3]
>>> my_list.append(4)
>>> my_list
[1, 2, 3, 4]
>>> my_list.insert(2, 'dog')
>>> my_list
[1, 2, 'dog', 3, 4]
>>> my_list.extend([5, 6])
>>> my_list
```

```
[1, 2, 'dog', 3, 4, 5, 6]
>>> my_list.append([7, 8])
>>> my_list
[1, 2, 'dog', 3, 4, 5, 6, [7, 8]]
>>> my_list.pop(2)
'dog'
>>> my_list
[1, 2, 3, 4, 5, 6, [7, 8]]
>>> my_list.reverse()
>>> my_list
[[7, 8], 6, 5, 4, 3, 2, 1]
```

Lists + functional programming

```
>>> def divisible_by_2(x):
...     return x % 2 == 0
...
>>>
>>> def cube(x):
...     return x ** 3
...
>>>
>>> numbers = [1, 2, 3, 4, 6, 31]
>>>
>>> filter(divisible_by_2, numbers)
>>> [2, 4, 6]
>>>
>>> map(cube, numbers)
>>> [1, 8, 27, 64, 216, 29791]
```

Generators

```
def print_5_lines(filename):
    """ Prints the first 5 lines of the given file. """
    my_file = open(filename)     # my_file is a generator expression
    for i in range(5):
        line = my_file.next()
        print line
```

This is useful for gigantic files, such as 100MB system logs.

Filesystem tools

List all the .log files in a directory, including all subdirectories.

```
import os
import os.path

for dirpath, dirnames, filenames in os.walk("."):
    for filename in [f for f in filenames if f.endswith(".log")]:
        print os.path.join(dirpath, filename)
```

Code taken from http://stackoverflow.com/questions/954504/how-to-get-files-in-a-directory-including-all-subdirectories

Dealing with tarfiles

Open and read nested tarfiles, without having to write the extracted files to disk.

```
>>> import tarfile
>>> baz = tarfile.open('baz.tgz')
>>> bar = tarfile.open(fileobj=baz.extractfile('bar.tgz'))
>>> bar.extractfile('bar/baz.txt').read()
'This is bar/baz.txt.\n'
```

From http://stackoverflow.com/questions/3293809/how-to-walk-a-tar-gz-file-that-contains-zip-files-without-extraction

CHAPTER 2

Heroku Case Study

Info Heroku and me

Author Daniel Greenfeld (http://pydanny.github.com)

Contents:

Code Samples

```
# Some helpful utility commands.

all: deploy

migrate:
    git push heroku master
    python project/manage.py syncdb --noinput --settings=settings.prod
    python project/manage.py migrate --noinput --settings=settings.prod

deploy:
    heroku pgbackups:capture --expire
    git push heroku master
    python project/manage.py syncdb --noinput --settings=settings.prod
    python project/manage.py migrate --noinput --settings=settings.prod
    python project/manage.py collectstatic --noinput --settings=settings.prod

style:
    python project/manage.py make_bookmarklet prod --settings=settings.dev
    git add project/static/js/bookmarklet.js
    git commit -m "[CN auto-commit]: rendered bookmarklet.js for production"
    git push heroku master
    python project/manage.py collectstatic --noinput --settings=settings.prod
    python project/manage.py make_bookmarklet dev --settings=settings.dev
    git add project/static/js/bookmarklet.js
    git commit -m "[CN auto-commit]: rendered bookmarklet.js back to development"
```

CHAPTER 3

List of Dictionaries

```
collection = []
document = {
        '_objectId': ObjectId('4f844e916c97c1000c000003'),
        'username': 'pydanny',
        'fiancee': 'audreyr'
    }
collection = [document, ]
collection = [document]  # JavaScript version. :-)
```

CHAPTER 4

MongoKit Model Example

```
from mongokit import Document, Connection

connection = Connection()

@connection.register
class Review(Document):
    structure = {
            'title':unicode,
            'body':unicode,
            'author':unicode,
            'created':datetime.datetime,
            'rating':int
    }
    required_fields = ['title', 'author', 'created']
    default_values = {'rating': 0, 'created': datetime.utcnow}
```

CHAPTER 5

MongoKit Query Example

```
>>> from mongokit import Connection
>>> connection = Connection()
>>> for review in connection.Review.find({'rating': 3}):
...     review['title']
```

CHAPTER 6

MongoEngine Model Example

```
import mongoengine as me

class Review(me.Document):

    title = me.StringField()
    body = me.StringField()
    author = me.StringField()
    created = me.DateTimeField(default=datetime.utcnow)
    rating = me.IntField()
```

CHAPTER 7

MongoEngine Query Example

```
>>> from reviews.models import Review
>>> for review in Review.objects.all():
...     review.title
```

CHAPTER 8

PyMongo Model Example

```
>>> from pymongo import Connection
>>> connection = Connection()
>>> my_data = {'rating': 3, 'title': 'I like ice cream'}
>>> connection.reviews.insert(my_data)
>>> your_data = {'rating': 3, 'name': 'You like ice cream'}
>>> connection.reviews.insert(your_data)
```

CHAPTER 9

PyMongo Query Example

```
[
    {'rating': 3,'title': 'I like ice cream'},
    {'rating': 3, 'name': 'You like ice cream'}
]
```

```
>>> connection = pymongo.Connection()
>>> db = connection.db
>>> for review in db.reviews.find({'rating': 3}):
...     review['title']
>>> for review in db.reviews.find(
...         {"title": {"$regex": "ice cream"} }
...         ):
...     review['title']
```