python-style-guide

python-guide

Published
with GitBook

目錄

Python 代码、单元测试和项目规范

> 代码是写给人看的, 顺带能在计算机上运行
>
> -- 《*SICP*》

目的

关于 python 项目，尤其是 web 项目，并没有一个比较完整（包含代码、测试、项目）的开源规范。所以我决定自己整理一个，作为大家制定python代码规范的参考，也可以直接拿来用。

在线阅读和下载

在线阅读 / PDF / EPUB / MOBI

参与项目

本书还需要不断完善和编辑，欢迎直接评论或者发起issue，也欢迎pull requests。

待完成

- 翻译awesome-python
- pylint
- virtualenv

License

python代码规范

本代码规范基于PEP8, 在PEP8的基础上做了一些改动。

简明概述

编码

- 如无特殊情况, 文件一律使用UTF-8编码
- 如无特殊情况, 文件头部必须加入 `#-*-coding:utf-8-*-` 标识

代码格式

缩进

- 统一使用4个空格进行缩进

行宽

每行代码尽量不超过80个字符(在特殊情况下可以略微超过80，但最长不得超过120)

理由：

- 这在查看side-by-side的diff时很有帮助
- 方便在控制台下查看代码
- 太长可能是设计有缺陷

引号

简单说，自然语言使用双引号，机器标示使用单引号，因此 代码里 多数应该使用 单引号

- 自然语言 使用双引号 `"..."`
  例如错误信息；很多情况还是unicode，使用 `u"你好世界"`
- 机器标识 使用单引号 `'...'` 例如dict里的key
- 正则表达式 使用原生的双引号 `r"..."`
- 文档字符串***(docstring)*** 使用三个双引号 `"""....."""`

空行

- 模块级函数和类定义之间空两行；
- 类成员函数之间空一行；

```
class A:

    def __init__(self):
        pass

    def hello(self):
        pass


def main():
    pass
```

- 可以使用多个空行分隔多组相关的函数
- 函数中可以使用空行分隔出逻辑相关的代码

编码

- 文件使用UTF-8编码
- 文件头部加入 `#-*-conding:utf-8-*-` 标识

import语句

- import语句应该分行书写

```
# 正确的写法
import os
import sys

# 不推荐的写法
import sys,os

# 正确的写法
from subprocess import Popen, PIPE
```

- import语句应该使用 **absolute** import

```
# 正确的写法
from foo.bar import Bar

# 不推荐的写法
from ..bar import Bar
```

- import语句应该放在文件头部，置于模块说明及docstring之后，于全局变量之前；
- import语句应该按照顺序排列，每组之间用一个空行分隔

```
import os
import sys

import msgpack
import zmq

import foo
```

- 导入其他模块的类定义时，可以使用相对导入

```
from myclass import MyClass
```

- 如果发生命名冲突，则可使用命名空间

```
import bar
import foo.bar

bar.Bar()
foo.bar.Bar()
```

空格

- 在二元运算符两边各空一格 `[=,-,+=,==,>,in,is not, and]`：

```
# 正确的写法
i = i + 1
submitted += 1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a + b) * (a - b)

# 不推荐的写法
i=i+1
submitted +=1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

- 函数的参数列表中，`,` 之后要有空格

```
# 正确的写法
def complex(real, imag):
    pass

# 不推荐的写法
def complex(real,imag):
    pass
```

- 函数的参数列表中，默认值等号两边不要添加空格

```
# 正确的写法
def complex(real, imag=0.0):
    pass

# 不推荐的写法
def complex(real, imag = 0.0):
    pass
```

- 左括号之后，右括号之前不要加多余的空格

```
# 正确的写法
spam(ham[1], {eggs: 2})

# 不推荐的写法
spam( ham[1], { eggs : 2 } )
```

- 字典对象的左括号之前不要多余的空格

```
# 正确的写法
dict['key'] = list[index]

# 不推荐的写法
dict ['key'] = list [index]
```

- 不要为对齐赋值语句而使用的额外空格

```
# 正确的写法
x = 1
y = 2
long_variable = 3

# 不推荐的写法
x             = 1
y             = 2
long_variable = 3
```

换行

Python支持括号内的换行。这时有两种情况。

1) 第二行缩进到括号的起始处

```
foo = long_function_name(var_one, var_two,
                         var_three, var_four)
```

2) 第二行缩进4个空格，适用于起始括号就换行的情形

```
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)
```

使用反斜杠 `\` 换行，二元运算符 `+` `.` 等应出现在行末；长字符串也可以用此法换行

```
session.query(MyTable).\
        filter_by(id=1).\
        one()

print 'Hello, '\
      '%s %s!' %\
      ('Harry', 'Potter')
```

禁止复合语句，即一行中包含多个语句：

```
# 正确的写法
do_first()
do_second()
do_third()

# 不推荐的写法
do_first();do_second();do_third();
```

`if/for/while` 一定要换行：

```
# 正确的写法
if foo == 'blah':
    do_blah_thing()

# 不推荐的写法
if foo == 'blah': do_blash_thing()
```

注释

块注释

“#”号后空一格，段落件用空行分开（同样需要“#”号）

```
# 块注释
# 块注释
#
# 块注释
# 块注释
```

行注释

至少使用两个空格和语句分开，注意不要使用无意义的注释

```
# 正确的写法
x = x + 1  # 边框加粗一个像素

# 不推荐的写法(无意义的注释)
x = x + 1 # x加1
```

docstring

docstring的规范在 PEP 257 中有详细描述，其中最其本的两点：

1. 所有的公共模块、函数、类、方法，都应该写docstring。私有方法不一定需要，但应该在def后提供一个块注释来说明。
2. docstring的结束"""应该独占一行，除非此docstring只有一行。

```
"""Return a foobar
Optional plotz says to frobnicate the bizbaz first.
"""

"""Oneline docstring"""
```

命名规范

- 应避免使用小写字母l(L)，大写字母O(o)或I(i)单独作为一个变量的名称，以区分数字1和0
- 包和模块使用全小写命名，尽量不要使用下划线
- 类名使用CamelCase命名风格，内部类可用一个下划线开头

- 函数使用下划线分隔的小写命名
- 当参数名称和Python保留字冲突，可在最后添加一个下划线，而不是使用缩写或自造的词
- 常量使用以下划线分隔的大写命名

```
MAX_OVERFLOW = 100

Class FooBar:

    def foo_bar(self, print_):
        print(print_)
```

注释和文档

代码注释和文档(docstring)的规范, 参考并综合了PEP257、 Google Python Style Guide 和 Numpy的文档标准.

注释

- 单行注释和多行注释, 请参考本章第一小节
- 在代码的关键部分(或比较复杂的地方), 能写注释的要尽量写注释
- 比较重要的注释段, 使用多个等号隔开, 可以更加醒目, 突出重要性

```
app = create_app(name, options)


# ====================================
# 请勿在此处添加 get post等app路由行为 !!!
# ====================================


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

文档注释（Docstring）

作为文档的Docstring一般出现在模块头部、函数和类的头部，这样在python中可以通过对象的__doc__对象获取文档. 编辑器和IDE也可以根据Docstring给出自动提示.

- 文档注释以 """ 开头和结尾, 首行不换行, 如有多行, 末行必需换行, 以下是Google的docstring风格示例

```
# -*- coding: utf-8 -*-
"""Example docstrings.

This module demonstrates documentation as specified by the `Google Python
Style Guide`_. Docstrings may extend over multiple lines. Sections are created
with a section header and a colon followed by a block of indented text.

Example:
    Examples can be given using either the ``Example`` or ``Examples``
    sections. Sections support any reStructuredText formatting, including
    literal blocks::

        $ python example_google.py

Section breaks are created by resuming unindented text. Section breaks
are also implicitly created anytime a new section starts.
"""
```

- 不要在文档注释复制函数定义原型, 而是具体描述其具体内容, 解释具体参数和返回值等

```
#  不推荐的写法(不要写函数原型等废话)
def function(a, b):
    """function(a, b) -> list"""
    ... ...


#  正确的写法
def function(a, b):
    """计算并返回a到b范围内数据的平均值"""
    ... ...
```

- 对函数参数、返回值等的说明采用numpy标准, 如下所示

```
def func(arg1, arg2):
    """在这里写函数的一句话总结(如: 计算平均值).

    这里是具体描述.

    参数
    ----------
    arg1 : int
        arg1的具体描述
    arg2 : int
        arg2的具体描述

    返回值
    -------
    int
        返回值的具体描述

    参看
    --------
    otherfunc : 其它关联函数等...

    示例
    --------
    示例使用doctest格式, 在`>>>`后的代码可以被文档测试工具作为测试用例自动运行

    >>> a=[1,2,3]
    >>> print [x + 3 for x in a]
    [4, 5, 6]
    """
```

- 文档注释不限于中英文, 但不要中英文混用
- 文档注释不是越长越好, 通常一两句话能把情况说清楚即可
- 模块、公有类、公有方法, 能写文档注释的, 应该尽量写文档注释

命名

应该避免的名称

以下命名应该尽量避免

- 单字符名称, 除了计数器和迭代器.

```
if __name__ == '__main__':
    # 不推荐的写法
    # 尽量避免单字符变量名
    s = "hello world!"
```

- 包/模块名中的连字符(-)

```
# 错误的包名
# 引用文件 html-parser.py
import html-parser

# 正确的写法
# 文件名应为 html_parser.py
import html_parser
```

- 双下划线开头并结尾的名称(Python保留, 例如__init__)
- 应避免使用小写字母l(L)，大写字母O(o)或I(i)单独作为一个变量的名称，以区分数字1和0

```
if __name__ == '__main__':
    # 不推荐的写法
    # 尽量避免l、O 等容易混淆的字母
    l = 1
    O = 0
    l = (O + 1)*l
```

- 当参数名称和Python保留字冲突，可在最后添加一个下划线，而不是使用缩写或自造的词

```
# 如果变量名和python保留字冲突，则在末尾添加下划线
# 切记不要自己造词，或者使用缩写
def print_():
    ... ...

if __name__ == '__main__':
    str_ = "hello world!"
    print_(str_)
```

命名约定

模块

- 模块尽量使用小写命名，首字母保持小写，尽量不要用下划线(除非多个单词，且数量不多的情况)

```
# 正确的模块名
import decoder
import html_parser

# 不推荐的模块名
import Decoder
```

类名

- 类名使用驼峰(CamelCase)命名风格，首字母大写，私有类可用一个下划线开头

```
class Farm():
    pass

class AnimalFarm(Farm):
    pass

class _PrivateFarm(Farm):
    pass
```

- 将相关的类和顶级函数放在同一个模块里. 不像Java, 没必要限制一个类一个模块.

函数

- 函数名一律小写，如有多个单词，用下划线隔开

```
def run():
    pass

def run_with_env():
    pass
```

- 私有函数在函数前加一个下划线_

```
class Person():

    def _private_func():
        pass
```

变量名

- 变量名尽量小写, 如有多个单词，用下划线隔开

```
if __name__ == '__main__':
    count = 0
    school_name = ''
```

- 常量采用全大写，如有多个单词，使用下划线隔开

```
MAX_CLIENT = 100
MAX_CONNECTION = 1000
CONNECTION_TIMEOUT = 600
```

程序设计规范

避免'裸代码'

- 尽量不要直接将代码写在模块的顶层中，在执行主程序前应该总是检查 `if __name__ == '__main__'` ，这样当模块被导入时主程序就不会被执行.

```
# 不推荐的写法(裸代码)
do_something()

# 正确的写法
def main():
    do_something()

if __name__ == '__main__':
    main()
```

所有的顶级代码在模块导入时都会被执行. 要小心不要去调用函数, 创建对象, 或者执行那些不应该在使用pydoc时执行的操作.

字符串

- 尽量不要用 `+` 号拼接字符串, 使用 `%s` 模板或 `join` 拼接

```
# 不推荐的写法
print("Spam" + " eggs" + " and" + " spam")
# 正确写法, 使用 join
print(" ".join(["Spam","eggs","and","spam"]))
# 正确写法, 使用 %s 模板
print("%s %s %s %s" % ("Spam", "eggs", "and", "spam"))
```

列表和字典

- 在不复杂的情况下, 尽量多用列表生成式, 可以使代码更加清晰

注意: 复杂情况下不适用, 列表生成式过复杂反而会导致阅读和调试困难

```
# 不推荐的写法
items = []
for item in directory:
    items.append(item)

# 正确的写法
items = [item for item in directory]
```

- 尽量使用map和filter等内置函数, 而不是自己去写循环

```
numbers = [1, 2, 6, 9, 10 ...]

# 不推荐的写法
def even(numbers):
    evens = []
    for number in numbers:
        if number % 2 == 0:
            evens.append(number)
    return even

# 正确的写法
def even(numbers):
    return filter(lambda x: x%2 == 0, numbers)
```

正则表达式

- 正则表达式前一律加r

```
regex_phone = re.compile(r'(13\d|14[57]|15[^4,\D]))$')
```

- 正则表达式使用前尽量先编译好

```
# 不推荐的写法, 不要使用未编译的正则
result = re.match(r'(^(13\d|14[57]|15[^4,\D]))$', my_phone)

# 正确的写法, 使用前先编译
regex_phone = re.compile(r'(13\d|14[57]|15[^4,\D]))$')

if __name__ == '__main__':
    result = regex_phone.match(my_phone)
```

单元测试规范

关于单元测试的规范

> 本文参考了 *Div Into Python3* 以及 stackoverflow 上的一些答案

什么是单元测试

单元测试是对软件的基本组成单位进行的测试，目的是检验软件基本组成单位的正确性。

单元测试是整个以测试为中心的开发策略中的一个重要部分。编写单元测试应该安排在项目的早期，同时要让它随同代码及需求变更一起更新。

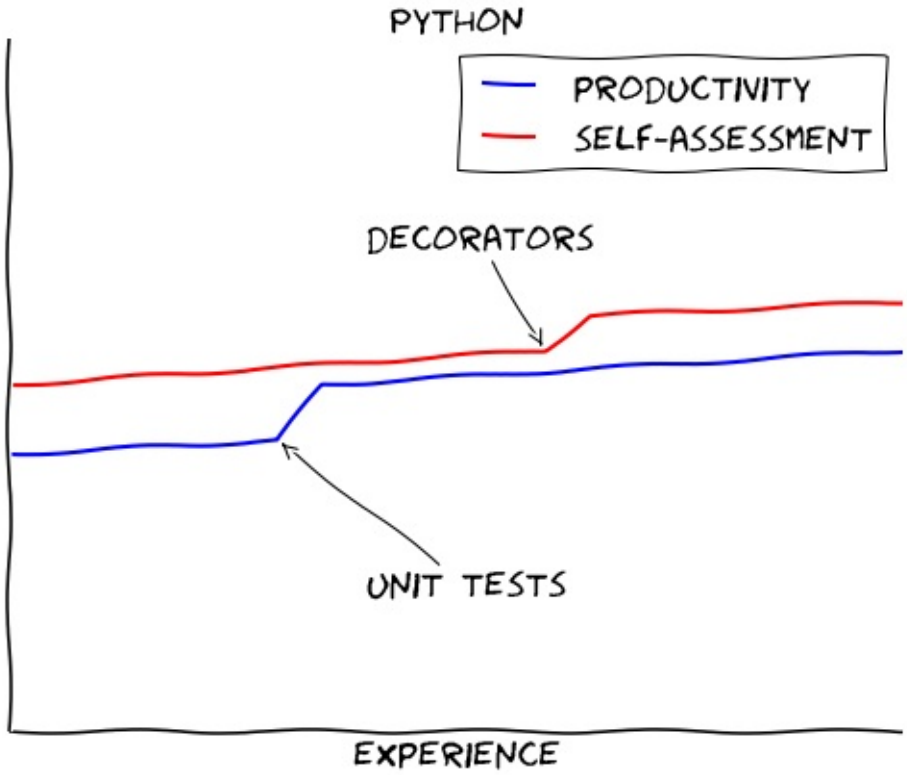
当然，任何时间编写单元测试都是有好处的。

单元测试的优点

- 在编写代码之前，通过编写单元测试来强迫你使用有用的方式细化你的需求。
- 在编写代码时，单元测试可以使你避免过度编码。当所有测试用例通过时，实现的方法就完成了。
- 重构代码时，单元测试用例有助于证明新版本的代码跟老版本功能是一致的。
- 在维护代码期间，如果有人对你大喊：你最新的代码修改破坏了原有代码的状态，那么此时单元测试可以帮助你反驳（“先生，所有单元测试用例通过了我才提交代码的...”）。
- 在团队编码中，缜密的测试套件可以降低你的代码影响别人代码的机会，这是因为你需要优先执行别人的单元测试用例。（我曾经在代码冲刺见过这种实践。一个团队把任务分解，每个人领取其中一小部分任务，同时为其编写单元测试;然后，团队相互分享他们的单元测试用例。这样，所有人都可以在编码过程中提前发现谁的代码与其他人的不可以良好工作。）

Python 中的单元测试

这是一张很知名的玩笑式的Python学习曲线：



它确实说明了一些问题，总体上Python程序员的自我评估一直高于实际的产出，但是写了测试之后会提高生产力，掌握了装饰(bi)器之后，以为自己更牛×了，实际上并没有，因为装饰器不过是一个卡新手的语法糖。

Python 在Web方向上被国内很多中小型公司大规模使用，但是由于追求所谓的产品快速上线和"敏捷开发"，很多创业公司盲目学习 Facebook 之类的公司不配备测试职位，却没有一个优秀的团队来把测试工作顺带完成，导致很多项目不重视测试，甚至不写单元测试也是常有的事。

单元测试的书写规范

一个单元测试样例的规则

- 完全自动运行，而不需要人工干预。单元测试几乎是全自动的。
- 自主判断被测试的方法是通过还是失败，而不需要人工解释结果。
- 独立运行，而不依赖其它测试用例（即使测试的是同样的方法）。即，每一个测试用例都是一个孤岛。
- 每一个独立的测试都有它自己的不含参数及没有返回值的方法。如果方法不抛出异常而正常退出则认为测试通过;否则，测试失败。

测试文件的位置及命名规则

1. 为了编写测试用例，首先使该测试用例类成为unittest 模块的TestCase 类的子类。TestCase 提供了很多你可以用于测试特定条件的测试用例的有用的方法。
2. 所有测试样例放在项目根目录下的 test 文件夹中，如果你的所有测试都在一个文件中，也可以省略文件夹，直接把这个文件放在根目录下，命名为 `test_your_project.py` (例如 `test_requests.py`)。
3. 如前一个测试用例，创建一个继承于 unittest.TestCase 的类。你可以在每个类中实现多个测试（正如你在本节中将会看到的一样），但是我却选择了创建一个新类，因为该测试与上一个有点不同。这样，我们可以把正常输入的测试跟非法输入的测试分别放入不同的两个类中。
4. 测试本身是类一个方法，并且该方法以 test 开头命名。
5. 当你改变条件的时候，要确保同步更新那些提示错误信息的可读字符串。unittest 框架并不关心这些，但是如果你的代码抛出描述不正确的异常信息的话会使得手工调试代码变得困难。
6. 出现频率较多和不明朗的方法，需要加注释，比如 `test_string` 如果不加注释很难让人读懂测试的是哪一个 string。

我们以bottle的测试为例：

```
import unittest
from tools import warn
from bottle import MakoTemplate, mako_template, mako_view, touni

class TestMakoTemplate(unittest.TestCase):
    def test_string(self):
        """ Templates: Mako string"""
        t = MakoTemplate('start ${var} end').render(var='var')
        self.assertEqual('start var end', t)

    def test_file(self):
        """ Templates: Mako file"""
        t = MakoTemplate(name='./views/mako_simple.tpl').render(var='var')
        self.assertEqual('start var end\n', t)

    # ...
```

单元测试框架的选择

选择框架之前首先要确认一下我们的选项概念和她们之间的区别。

PyUnit

PyUnit（The Python unit testing framework) 是 Kent Beck 和 Erich Gamma 所设计的 JUnit 的 Python 版本。从 Python 2.1 版本后，PyUnit 成为 Python 标准库的一部分。

这就是我们日常使用的 unittest 。它太过优秀了，以至于我们几乎没有必要再用别的测试框架了。

下面这个例子，是bottle框架下测试首页的简单例子，首先是bottle的代码：

```
#!/usr/bin/env python
# encoding: utf-8
import bottle


@bottle.route('/')
def index():
    return 'Hi!'


if __name__ == '__main__':
    bottle.run()
```

这是单元测试的代码：

```
#!/usr/bin/env python
# encoding: utf-8
import mywebapp
import unittest


class TestBottle(unittest.TestCase):
    def test_webapp_index(self):
        assert mywebapp.index() == 'Hi!'


if __name__ == '__main__':
    unittest.main()
```

直接运行这个文件就可以了，可以看到在普通模式，和verbosity模式下的返回结果:

普通模式:

```
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

verbosity模式:

```
test_webapp_index (__main__.TestBottle) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

表示我们的测试都通过了。

doctest

doctest 也是 Python 标准库的一部分。

doctest 可以用来写一些比较简单的测试，也可以用来检查文档和注释能否解释当前版本的代码。

pytest

pytest 也是一个很流行的测试框架，是很多公司的首选。

看一份邮件流吧，一些工程师阐述了他们为什么会选择使用pytest: 我是邮件链接

nose

nose 并不是一个真正的测试框架，它只是一个很优秀的测试执行器，是unittest的拓展。

它的官方也是这样介绍自己的：nose extends unittest to make testing easier.

nose 可以运行由PyUnit(unittest), doctest，pytest创建的测试。

项目规范

关于python项目（尤其是web项目）的规范

包和依赖

1. 如无特殊情况，项目的包管理器一律使用**pip**
2. 项目根目录下必须包含**requirements.txt**文件，作为项目依赖的显示声明文件
3. 所有依赖都必须显示声明，禁止隐式依赖，例如：

   ```
   # 错误，依赖的wget在别的系统环境下很可能不存在
   def download(url):
       os.system('wget %s' % url)
   ```

   以上代码隐式依赖了wget，但很有可能开发环境中有wget，而在部署环境下根本不存在， 这样部署app时就很可能出错！
4. 依赖声明文件中不应该包含项目中没有用到的依赖
5. 建议在开发项目时使用virtualenv做依赖隔离，便于使用pip freeze自动生成

   ```
   # 自动生成requirement.txt示例
   $ pip freeze > requirements.txt
   ```

静态文件

> 如无特殊说明，本文中的静态文件是指web项目中的图片、css、js等静态资源而言（有cdn的需求）

1. 为了便于部署时使用cdn加速，静态文件应该统一存放于项目根目录下的 `static` 文件夹中
2. 为了便于统一管理，favicon.ico文件也应该放置于static文件夹中， 在html中使用link方式引用

```
<!DOCTYPE html>
<html>
<head>
<title>标题</title>
<link rel="shortcut icon" href="http://www.xxx.com/图标的地址.../favicon.ico">
</head>
... ...
```

3. 引用静态文件时，不要用绝对路径，可以使用django的static模板标签，或flask的url_for自动生成
   - django 静态文件引用示例

```
{% load staticfiles %}
<img src="{% static "my_app/myexample.jpg" %}" alt="My image"/>
```

   - flask 静态文件引用示例

```
<img src="{% url_for "my_app/myexample.jpg" %}" alt="My image"/>
```

   - 不恰当的静态文件引用方式

```
<img src="/static/img/pic1.jpg" alt="My image"/>
```

配置文件和数据

1. 如框架无特殊规定，配置文件应放置于项目根目录下的 `config` 文件夹中
2. 配置文件在部署、预发布、生产环境、开发环境等环境中会有很大差异，因此请不要将配置文件在上传到**git**、**svn**等版本库中， 而是建议在版本库中上传一个配置的示例文件（如：config.example）
3. 上传到版本库中配置示例文件不允许出现密码、证书、**token**等敏感信息
4. 数据和程序应该尽量分离，不要将数据写在代码中，需要持久化存储数据必须使用数据库

项目目录结构示例

以下是一个web项目的目录结构示例

```
.project_root/                          项目根目录
|
├── app/                                app代码等(不限定，可根据实际情况命名和确定结构)
|   ├── .../
|   ├── .../
|   └── ...
├── config/                             配置文件夹
|   ├── config.ini                      配置文件（应该被版本库ignore掉）
|   └── config.ini.example              配置文件示例
├── static/                             静态文件夹
|   ├── css/
|   ├── img/
|   ├── js/
|   └── favicon.ico                     网站图标
├── README.md                           项目说明文件
├── requrements.txt                     项目依赖文件
├── TODO.md                             待完成项目
└── .gitignore                          版本库ignore文件
```

优雅之道

> 关于工具，一个最重要的，也是最不易察觉的方面是，工具对使用此工具的人的习惯的潜移默化的影响。 如果这个工具是一门程序语言，不管我们是否喜欢它，它都会影响我们的思维惯式。
>
> -- *Edsger Dijkstra*

Python是一门优雅的语言，使用Python就意味着要用Python的思维方式。 实际上，很多人还用c语言的思维写Python, 这样写出来的代码读起来很费劲，而且效率也不高。 为了提高效率，本章介绍了一些python的设计哲学、设计原则、高效率的工具和轮子等。

Python的哲学

在python命令行中输入import this， 就会打印出一段‘The Zen of Python’的描述:

> 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!

这就是python的设计哲学，即’python之禅‘，其中文翻译如下：

> python之禅, by Tim Peters
>
> 优美胜于丑陋
> Beautiful is better than ugly.
> 明了胜于晦涩
> Explicit is better than implicit.
> 简单胜过复杂
> Simple is better than complex.
> 复杂胜过凌乱
> Complex is better than complicated.
> 扁平胜于嵌套
> Flat is better than nested.
> 间隔胜于紧凑
> parse is better than dense.
> 可读性很重要
> Readability counts.
> 即使假借特例的实用性之名，也不可违背这些规则
> Special cases aren't special enough to break the rules.
> 虽然实用性比纯粹性重要
> Although practicality beats purity.
> 不要包容所有错误（精准地捕获异常）
> Errors should never pass silently.
> 除非你确定需要这样做
> Unless explicitly silenced.
> 当存在多种可能时，不要尝试去猜测
> In the face of ambiguity, refuse the temptation to guess.
> 而是尽量找一种，最好是唯一种明显的解决方案
> There should be one-- and preferably only one --obvious way to do it.
> 虽然这并不容易，因为你不是 Python 之父
> Although that way may not be obvious at first unless you're Dutch.
> 做也许好过不做
> Now is better than never.
> 但不假思索就动手还不如不做
> Although never is often better than *right* now.
> 如果你无法向人描述你的实现，那肯定不是一个好方案
> If the implementation is hard to explain, it's a bad idea.
> 如果你很容易向人解释你的实现，那么它很可能是个好方案
> If the implementation is easy to explain, it may be a good idea.
> 命名空间是一种绝妙的理念，应当多加利用
> Namespaces are one honking great idea -- let's do more of those!

如果说语言是一种工具，那么毫无疑问，不同工具会对思维方式产生不同影响, 也许python程序员大部分有极简主义倾向，讨厌复杂的东西，他们更加关注如何更快地get things done.

python作为一种优雅的工具，有一些独到的优点：

```
- 简洁      更少的代码，  更多的事情
- 库比较多      大量现成的轮子，更快的get things done
- 语法清晰，可读性高      做一种事情通常只有一种最好的方法
```

我们应该充分运用这些优点，发挥更高的生产力，接下来的几节将描述致力于简洁性的DRY和KISS原则，以及介绍一些python好用的工具和轮子.

DRY原则 -- 不要重复你自己

Don't repeat yourself（不要重复你自己，简称DRY）是面向对象编程中的基本原则，程序员的行事准则。 旨在软件开发中，减少重复的信息。

DRY的原则是——系统中的每一部分，都必须有一个单一的、明确的、权威的代表——指的是（由人编写而非机器生成的）代码和测试所构成的系统，必须能够表达所应表达的内容，但是不能含有任何重复代码。 当DRY原则被成功应用时，一个系统中任何单个元素的修改都不需要与其逻辑无关的其他元素发生改变。此外，与之逻辑上相关的其他元素的变化均为可预见的、均匀的，并如此保持同步。

但过度的强调DRY，以及过早优化都是不对的，我们应该尽量避免过度抽象.有时候一定程度的冗余反而是好事.

关于抽象的原则，可以参看阮一峰的这篇文章

KISS -- keep it simple & small

KISS原则是英语keep it simple & small, 或Keep It Simple, Stupid 的首字母缩略字。KISS原则是指在设计当中应当注重简约的原则。

KISS原则在设计上可能最被推崇的，在家装设计，界面设计，操作设计上，复杂的东西越来越被众人所BS了，而简单的东西越来越被人所认可，比如这些UI的设计和我们中国网页（尤其是新浪的网页）者是负面的例子。“宜家”（IKEA）简约、效率的家居设计、生产思路；“微软”（Microsoft）“所见即所得”的理念；“谷歌”（Google)简约、直接的商业风格，无一例外的遵循了“kiss”原则，也正是“kiss”原则，成就了这些看似神奇的商业经典。而苹果公司的iPhone/iPad将这个原则实践到了极至。

把一个事情搞复杂是一件简单的事，但要把一个复杂的事变简单，这是一件复杂的事。

简单的东西往往更容易维护，更加稳定，也更加容易做好，这也是Unix的设计哲学（Less is more）：各模块只做好自己该做的事情.

在软件设计的时候，我们也需要想清楚，某些东西是否真的需要， 也许You aren't gonna need it.

虚拟环境--virtualenv

Lint工具--pylint

Python的轮子 -- 常用的包、库、软件

一些比较好用的 Python 框架、库、软件等的合集, 基于awesome-python.

    - HTML处理
    - Web爬虫
    - Web内容解析
    - 数据验证
    - 管理界面
    - 静态网站生成
    - 进程管理
    - 并发和并行
    - 网络
    - WebSocket处理
    - WSGI服务器
    - 远程调用服务
    - 加解密
    - 图形界面（GUI）
    - 游戏开发
    - 日志工具
    - 测试
    - 代码分析和优化
    - 调试工具
    - 科学算和数据分析
    - 数据可视化
    - 计算机图形学
    - 机器学习
    - 函数式编程
    - MapReduce
    - 运维开发工具
    - 任务调度
    - 使用其它语言扩展Python
    - 高性能
    - Windows工具
    - 网络虚拟化和SDN
    - 硬件
    - 兼容性
    - 开发者插件
    - IDEs
- Python资源
    - 网站
    - 网站
    - Twitter

Python环境管理

*Python*版本和虚拟环境管理.

- virtualenv - 最常用的Python虚拟环境管理工具.
- p - 非常简单的Python版本管理工具.
- pyenv - 简单的Python版本管理工具.
- PyRun - 一个单文件、免安装的Python运行环境.
- Vex - 在virtualenv中运行命令的工具.
- virtualenvwrapper - virtualenv的插件合集.

包管理工具

包和依赖管理工具.

- pip - 即Python Package Index, 最常用的Python包管理工具.
  - 官方网站 Python Package Index
- pip-tools - 保持pip的包最新的同步工具.
- conda - 跨平台的二进制包管理工具.
- Curdling - 命令行包管理工具.
- wheel - 标准Python打包工具（用来替代eggs）.

打包发布工具

打包发布和安装包制作工具.

- PyInstaller - 将Python程序转换为跨平台的可执行程序(客户无需安装python即可运行).
- dh-virtualenv - 将virtualenv环境打包成可发布的Debian软件包.
- Nuitka - 将python脚本、模块、包编译成可执行或可扩展模块.
- py2app - Python代码打包成 Mac OS X App 的工具.
- py2exe - Python转exe(Windows).
- pynsist - Windows安装包制作工具，会将Python一起打包.

交互式工具（REPL）

交互式*Python*执行工具 *(REPL)*.

- IPython - 强大，比较常用的Python交互shell.
- bpython – 一个便捷的Python交互工具.
- ptpython - 先进的Python交互工具.

文件

文件操作和*MIME*类型描述等.

- imghdr - (Python内置标准库) 判断图片类型的库.
- mimetypes - (Python内置标准库) 获取文件的MIME类型描述.
- path.py - os.path的便捷版本.
- pathlib - (Python3.4+ 内置标准库) 跨平台的文件路径库.
- python-magic - libmagic 的便捷版本.
- Unipath - 一个面向对象的文件和目录操作库.
- watchdog - 监视文件系统改动的库.

时间和日期

关于时间和日期的库.

- arrow - 更好用的时间日期库.
- Chronyk - Python3 时间日期解析库.
- dateutil - Python标准模块 datetime 的扩展.
- delorean - 处理跨时区、时间和日期的库.
- moment - 时间日期处理库，Moment.js的Python 版.
- PyTime - 易用的的时间处理库，把date/time/datetime当成string来处理.
- pytz - 世界时区的库. 使用 tz database.
- when.py - 用户友好的时间日期操作库.

文本处理

解析和操作普通文本的库.

- 通用
  - chardet - 兼容 Python 2/3 的字符编码检测.
  - difflib - (Python内置标准库) 差异计算工具.
  - esmre - 正则表达式加速器.
  - ftfy - Unicode文本乱码自动修复.
  - fuzzywuzzy - 文本模糊匹配(包含相似度计算).
  - Levenshtein - 字符串莱文斯坦距离和相似度计算.
  - pangu.py - 中日韩文分词库.
  - pyfiglet - figlet的Python实现.
  - shortuuid - 简洁，唯一，URL安全的UUID生成器.
  - unidecode - Unicode转ASCII.

    - uniout - 打印可读的字符串（自动消除转义）.
    - xpinyin - 将中文汉字转化为拼音.
- 字符串转义(Slugify)
    - awesome-slugify - 不转换unicode的slugify库.
    - python-slugify - slugify库，将会把unicode转成ASCII.
    - unicode-slugify - slugify库, 支持unicode.
- 解析器(Parser)
    - phonenumbers - 解析、格式化、存储和验证电话号码.
    - PLY - lex 和 yacc 解析工具的python实现.
    - Pygments - （编程语言）文本语法高亮工具.
    - pyparsing - 通用parser生成框架.
    - python-nameparser - 人名解析器（英文）.
    - python-user-agents - 浏览器user-agent解析器.
    - sqlparse - SQL语句解析器.

特定文本格式处理（word、excel、pdf、markdown等）

解析和处理特定文本格式的库.

- 通用
    - tablib - XLS, CSV, JSON, YAML处理.
- Office
    - Marmir - 把Python数据结构转化成表格.
    - openpyxl - 读写微软Excel 2010 xlsx/xlsm/xltx/xltm 文件.
    - python-docx - 读写微软Word 2007/2008 docx文件.
    - unoconv - LibreOffice/OpenOffice 文档格式转化.
    - XlsxWriter - 创建 .xlsx 文件的库.
    - xlwings - 读写 Excel.
    - xlwt / xlrd - 读写 Excel 文件.
    - relatorio - OpenDocument文件模版库.
- PDF
    - PDFMiner - 读取PDF文档.
    - PyPDF2 - 切分、合并、转换PDF文档
    - ReportLab - 快速创建PDF文档.
- Markdown
    - Mistune - 快速，功能齐全的Markdown解析器.
    - Python-Markdown - John Gruber的Markdown的Python实现.
- YAML
    - PyYAML - YAML的Python实现.

- CSV
  - csvkit - CSV转化和操作库.
- 压缩文件
  - unp - 压缩文件处理库（.tar, .gz, .zip, .7z, .bz, .rar 等）.

自然语言处理

自然语言处理库.

- NLTK - 最常用的Python自然语言处理平台.
- jieba - 中文分词.
- langid.py - 自动判断语言类型（可判别英文、中文等97种语言）.
- Pattern - 网络文本挖掘和分析.
- SnowNLP - 中文文本处理库(分词、词性、情感分析等).
- TextBlob - 便捷的NLP通用库.

文档工具

生成项目文档的工具.

- Sphinx - 功能强大的文档生成器，广泛用于Python社区的各种文档.
- MkDocs - 使用Markdown的文档生成器.
- pdoc - 自动为Python库生成文档.
- Pycco - "文艺编程"风格的文档生成器.

配置

存储和解析配置的库.

- config
- ConfigObj - INI配置文件处理.
- ConfigParser - (Python内置标准库) 解析INI文件.
- profig - 可解析多种配置文件格式.
- python-decouple - Strict separation of settings from code.

图片

Libraries for manipulating images.

- pillow - Pillow is the friendly PIL fork.
- hmap - Image histogram remapping.
- imgSeek - A project for searching a collection of images using visual similarity.
- nude.py - Nudity detection.
- pyBarcode - Create barcodes in Python without needing PIL.
- pygram - Instagram-like image filters.
- python-qrcode - A pure Python QR Code generator.
- Quads - Computer art based on quadtrees.
- scikit-image - A Python library for (scientific) image processing.
- thumbor - A smart imaging service. It enables on-demand crop, re-sizing and flipping of images.
- wand - Python bindings for MagickWand, C API for ImageMagick.

文字识别（OCR）

文字识别库, *OCR(Optical Character Recognition).*

- pyocr - A wrapper for Tesseract and Cuneiform.
- pytesseract - Another wrapper for Google Tesseract OCR.
- python-tesseract - A wrapper class for Google Tesseract OCR.

音频

Libraries for manipulating audio.

- audiolazy - Expressive Digital Signal Processing (DSP) package for Python.
- audioread - Cross-library (GStreamer + Core Audio + MAD + FFmpeg) audio decoding.
- beets - A music library manager and MusicBrainz tagger.
- dejavu - Audio fingerprinting and recognition.
- django-elastic-transcoder - Django + Amazon Elastic Transcoder.
- eyeD3 - A tool for working with audio files, specifically MP3 files containing ID3 metadata.
- id3reader - A Python module for reading MP3 meta data.
- m3u8 - A module for parsing m3u8 file.
- mutagen - A Python module to handle audio metadata.
- pydub - Manipulate audio with a simple and easy high level interface.
- pyechonest - Python client for the Echo Nest API.
- talkbox - A Python library for speech/signal processing.
- TimeSide - Open web audio processing framework.
- tinytag - A library for reading music meta data of MP3, OGG, FLAC and Wave files.

- mingus - An advanced music theory and notation package with MIDI file and playback support.

视频

Libraries for manipulating video and GIFs.

- moviepy - A module for script-based movie editing with many formats, including animated GIFs.
- scikit-video - Video processing routines for SciPy.

地理位置

Libraries for geocoding addresses and working with latitudes and longitudes.

- GeoDjango - A world-class geographic web framework.
- GeoIP - Python API for MaxMind GeoIP Legacy Database.
- geojson - Python bindings and utilities for GeoJSON.
- geopy - Python Geocoding Toolbox.
- pygeoip - Pure Python GeoIP API.
- django-countries - A Django app that provides country choices for use with forms, flag icons static files, and a country field for models.

HTTP

Libraries for working with HTTP.

- requests - HTTP Requests for Humans™.
- grequests - requests + gevent for asynchronous HTTP requests.
- httplib2 - Comprehensive HTTP client library.
- treq - Python requests like API built on top of Twisted's HTTP client.
- urllib3 - A HTTP library with thread-safe connection pooling, file post support, sanity friendly.

数据库

用*Python*写的数据库.

- pickleDB - A simple and lightweight key-value store for Python.

- PipelineDB - The Streaming SQL Database.
- TinyDB - A tiny, document-oriented database.
- ZODB - A native object database for Python. A key-value and object graph database.

数据库驱动（连接件）

连接和操作数据库.

- MySQL
  - mysql-python - Python默认的MySQL连接件.
  - mysqlclient - 支持Python 3，是mysql-python的一个分支.
  - PyMySQL - 纯Python MySQL连接件，兼容mysql-python.
  - oursql - 一个更好用的MySQL连接件.
- PostgreSQL
  - psycopg2 - 最常用的PostgreSQL连接件.
  - queries - A wrapper of the psycopg2 library for interacting with PostgreSQL.
  - txpostgres - Twisted based asynchronous driver for PostgreSQL.
- 其它关系型数据库
  - apsw - Another Python SQLite wrapper.
  - dataset - Store Python dicts in a database - works with SQLite, MySQL, and PostgreSQL.
  - pymssql - A simple database interface to Microsoft SQL Server.
- NoSQL数据库
  - cassandra-python-driver - Python driver for Cassandra.
  - HappyBase - A developer-friendly library for Apache HBase.
  - Plyvel - A fast and feature-rich Python interface to LevelDB.
  - py2neo - Python wrapper client for Neo4j's restful interface.
  - pycassa - Python Thrift driver for Cassandra.
  - PyMongo - The official Python client for MongoDB.
  - redis-py - The Redis Python Client.
  - telephus - Twisted based client for Cassandra.
  - txRedis - Twisted based client for Redis.

数据库对象关系模型映射（ORM）

Libraries that implement Object-Relational Mapping or data mapping techniques.

- 关系型数据库
  - Django Models - A part of Django.
  - SQLAlchemy - The Python SQL Toolkit and Object Relational Mapper.

        - awesome-sqlalchemy
    - Peewee - A small, expressive ORM.
    - PonyORM - ORM that provides a generator-oriented interface to SQL.
    - python-sql - Write SQL queries pythonically.
- NoSQL数据库
    - django-mongodb-engine - Django MongoDB Backend.
    - PynamoDB - A Pythonic interface for Amazon DynamoDB.
    - flywheel - Object mapper for Amazon DynamoDB.
    - MongoEngine - A Python Object-Document-Mapper for working with MongoDB.
    - hot-redis - Rich Python data types for Redis.
    - redisco - A Python Library for Simple Models and Containers Persisted in Redis.
- 其它
    - butterdb - A Python ORM for Google Drive Spreadsheets.

Web框架

Full stack web frameworks.

- Django - The most popular web framework in Python.
    - awesome-django
- Flask - A microframework for Python.
    - awesome-flask
- Pyramid - A small, fast, down-to-earth, open source Python web framework.
    - awesome-pyramid
- Bottle - A fast, simple and lightweight WSGI micro web-framework.
- CherryPy - A minimalist Python web framework, HTTP/1.1-compliant and WSGI thread-pooled.
- TurboGears - A microframework that can scale up to a full stack solution.
- web.py - A web framework for Python that is as simple as it is powerful.
- web2py - A full stack web framework and platform focused in the ease of use.

内容管理系统（CMS）

Content Management Systems.

- django-cms - An Open source enterprise CMS based on the Django.
- djedi-cms - A lightweight but yet powerful Django CMS with plugins, inline editing and performance in mind.
- FeinCMS - One of the most advanced Content Management Systems built on Django.
- Kotte - A high-level, Pythonic web application framework built on Pyramid.

- Mezzanine - A powerful, consistent, and flexible content management platform.
- Opps - A Django-based CMS for magazines, newspapers websites and portals with high-traffic.
- Plone - A CMS built on top of the open source application server Zope.
- Quokka - Flexible, extensible, small CMS powered by Flask and MongoDB.
- Wagtail - A Django content management system.
- Widgy - Last CMS framework, based on Django.

电子商务

Frameworks and libraries for e-commerce and payments.

- django-oscar - An open-source e-commerce framework for Django.
- django-shop - A Django based shop system.
- Cartridge - A shopping cart app built using the Mezzanine.
- shoop - An open source E-Commerce platform based on Django.
- alipay - Unofficial Alipay API for Python.
- merchant - A Django app to accept payments from various payment processors.
- money - Money class with optional CLDR-backed locale-aware formatting and an extensible currency exchange solution.
- python-currencies - Display money format and its filthy currencies.

RESTful API 框架

Libraries for developing RESTful APIs.

- Django
  - django-rest-framework - A powerful and flexible toolkit to build web APIs.
  - django-tastypie - Creating delicious APIs for Django apps.
  - django-formapi - Create JSON APIs with Django's form validation.
- Flask
  - flask-api - Browsable Web APIs for Flask.
  - flask-restful - Quickly building REST APIs for Flask.
  - flask-restless - Generating RESTful APIs for database models defined with SQLAlchemy.
  - flask-api-utils - Taking care of API representation and authentication for Flask.
  - eve - REST API framework powered by Flask, MongoDB and good intentions.
- Pyramid
  - cornice - A REST framework for Pyramid.
- 其它

    - falcon - A high-performance framework for building cloud APIs and web app backends.
    - sandman - Automated REST APIs for existing database-driven systems.
    - restless - Framework agnostic REST framework based on lessons learned from Tastypie.
    - ripozo - Quickly creating REST/HATEOAS/Hypermedia APIs.

授权（Authentication）

Libraries for implementing authentications schemes.

- OAuth
    - Authomatic - Simple but powerful framework agnostic authentication/authorization client.
    - django-allauth - Authentication app for Django that "just works."
    - django-oauth-toolkit - OAuth2 goodies for the Djangonauts.
    - django-oauth2-provider - Providing OAuth2 access to Django app.
    - Flask-OAuthlib - OAuth 1.0/a, 2.0 implementation of client and provider for Flask.
    - OAuthLib - A generic and thorough implementation of the OAuth request-signing logic.
    - python-oauth2 - A fully tested, abstract interface to creating OAuth clients and servers.
    - python-social-auth - An easy-to-setup social authentication mechanism.
    - rauth - A Python library for OAuth 1.0/a, 2.0, and Ofly.
    - sanction - A dead simple OAuth2 client implementation.
- 其它
    - jose - JavaScript Object Signing and Encryption draft implementation.
    - PyJWT - Implementation of the JSON Web Token draft 01.
    - python-jws - Implementation of JSON Web Signatures draft 02.
    - python-jwt - Module for generating and verifying JSON Web Tokens.

模板引擎

Libraries and tools for templating and lexing.

- Jinja2 - A modern and designer friendly templating language.
- Chameleon - An HTML/XML template engine. Modeled after ZPT, optimized for speed.
- Genshi - Python templating toolkit for generation of web-aware output.
- Mako - Hyperfast and lightweight templating for the Python platform.
- Spitfire - A very fast Python template compiler.

任务队列

Libraries for working with event and task queues.

- celery - An asynchronous task queue/job queue based on distributed message passing.
- huey - Little multi-threaded task queue.
- mrq - Mr. Queue - A distributed worker task queue in Python using Redis & gevent.
- rq - Simple job queues for Python.
- simpleq - A simple, infinitely scalable, Amazon SQS based queue.

搜索工具

Libraries and software for indexing and performing search queries on data.

- django-haystack - Modular search for Django.
- elasticsearch-py - The official low-level Python client for Elasticsearch.
- elasticsearch-dsl-py - The official high-level Python client for Elasticsearch.
- solrpy - A Python client for solr.
- Whoosh - A fast, pure Python search engine library.

Feed工具

Libraries for building user's activities.

- django-activity-stream - Generating generic activity streams from the actions on your site.
- Stream-Framework - Building newsfeed and notification systems using Cassandra and Redis.

网站资源管理（压缩、最小化等）

Tools for managing, compressing and minifying website assets.

- django-compressor - Compresses linked and inline JavaScript or CSS into a single cached file.
- django-storages - A collection of custom storage back ends for Django.
- fanstatic - Packages, optimizes, and serves static file dependencies as Python packages.
- File Conveyor - A daemon to detect and sync files to CDNs, S3 and FTP.
- Flask-Assets - Helps you integrate webassets into your Flask app.

- glue - Glue is a simple command line tool to generate CSS sprites.
- jinja-assets-compressor - A Jinja extension to compile and compress your assets.
- webassets - Bundles, optimizes, and manages unique cache-busting URLs for static resources.

缓存

Libraries for caching data.

- Beaker - A library for caching and sessions for use with web applications and stand-alone Python scripts and applications.
- django-cache-machine - Automatic caching and invalidation for Django models.
- django-cacheops - A slick ORM cache with automatic granular event-driven invalidation.
- django-viewlet - Render template parts with extended cache control.
- dogpile.cache - dogpile.cache is next generation replacement for Beaker made by same authors.
- HermesCache - Python caching library with tag-based invalidation and dogpile effect prevention.
- johnny-cache - A caching framework for django applications.
- pylibmc - A Python wrapper around the libmemcached interface.

电子邮件（email）

Libraries for sending and parsing email.

- django-celery-ses - Django email back end with AWS SES and Celery.
- envelopes - Mailing for human beings.
- flanker - A email address and Mime parsing library.
- imbox - Python IMAP for Humans.
- inbox.py - Python SMTP Server for Humans.
- inbox - The open source email toolkit.
- lamson - Pythonic SMTP Application Server.
- mailjet - Mailjet API implementation for batch mailing, statistics and more.
- marrow.mailer - High-performance extensible mail delivery framework.
- modoboa - A mail hosting and management platform including a modern and simplified Web UI.
- pyzmail - Compose, send and parse emails.
- Talon - Mailgun library to extract message quotations and signatures.

国际化（多语言）

Libraries for working with i18n.

- Babel - An internationalization library for Python.
- Korean - A library for Korean morphology.

URL处理

Libraries for parsing URLs.

- furl - A small Python library that makes manipulating URLs simple.
- purl - A simple, immutable URL class with a clean API for interrogation and manipulation.
- pyshorteners - A pure Python URL shortening lib.
- short_url - Python implementation for generating Tiny URL and bit.ly-like URLs.
- webargs - A friendly library for parsing HTTP request arguments, with built-in support for popular web frameworks, including Flask, Django, Bottle, Tornado, and Pyramid.

HTML处理

Libraries for working with HTML and XML.

- BeautifulSoup - Providing Pythonic idioms for iterating, searching, and modifying HTML or XML.
- bleach - A whitelist-based HTML sanitization and text linkification library.
- cssutils - A CSS library for Python.
- html5lib - A standards-compliant library for parsing and serializing HTML documents and fragments.
- lxml - A very fast, easy-to-use and versatile library for handling HTML and XML.
- MarkupSafe - Implements a XML/HTML/XHTML Markup safe string for Python.
- pyquery - A jQuery-like library for parsing HTML.
- untangle - Converts XML documents to Python objects for easy access.
- xhtml2pdf - HTML/CSS to PDF converter.
- xmltodict - Working with XML feel like you are working with JSON.

Web爬虫

Libraries for scraping websites.

- Scrapy - A fast high-level screen scraping and web crawling framework.
- cola - A distributed crawling framework.
- Demiurge - PyQuery-based scraping micro-framework.
- feedparser - Universal feed parser.
- Grab - Site scraping framework.
- MechanicalSoup - A Python library for automating interaction with websites.
- portia - Visual scraping for Scrapy.
- pyspider - A powerful spider system.
- RoboBrowser - A simple, Pythonic library for browsing the web without a standalone web browser.

Web内容解析

Libraries for extracting web contents.

- Haul - An Extensible Image Crawler.
- html2text - Convert HTML to Markdown-formatted text.
- lassie - Web Content Retrieval for Humans.
- micawber - A small library for extracting rich content from URLs.
- newspaper - News extraction, article extraction and content curation in Python.
- opengraph - A Python module to parse the Open Graph Protocol
- python-goose - HTML Content/Article Extractor.
- python-readability - Fast Python port of arc90's readability tool.
- sanitize - Bringing sanity to world of messed-up data.
- sumy - A module for automatic summarization of text documents and HTML pages.
- textract - Extract text from any document, Word, PowerPoint, PDFs, etc.

数据验证

Libraries for validating data. Used for forms in many cases.

- Cerberus - A mappings-validator with a variety of rules, normalization-features and simple customization that uses a pythonic schema-definition.
- colander - A system for validating and deserializing data obtained via XML, JSON, an HTML form post or any other equally simple data serialization.
- kmatch - A language for matching/validating/filtering Python dictionaries.
- schema - A library for validating Python data structures.
- Schematics - Data Structure Validation.
- valideer - Lightweight extensible data validation and adaptation library.
- voluptuous - A Python data validation library. It is primarily intended for validating data

coming into Python as JSON, YAML, etc.

管理界面

Libraries for administrative interfaces.

- Ajenti - The admin panel your servers deserve.
- django-suit - Alternative Django Admin-Interface (free only for Non-commercial use).
- django-xadmin - Drop-in replacement of Django admin comes with lots of goodies.
- flask-admin - Simple and extensible administrative interface framework for Flask.
- flower - Real-time monitor and web admin for Celery.
- Grappelli – A jazzy skin for the Django Admin-Interface.
- Wooey - A Django app which creates automatic web UIs for Python scripts.

静态网站生成

Static site generator is a software that takes some text + templates as input and produces HTML files on the output.

- Pelican - Uses Markdown or ReST for content and Jinja 2 for themes. Supports DVCS, Disqus. AGPL.
- Cactus – Static site generator for designers.
- Hyde - Jinja2-based static web site generator.
- Nikola - A static website and blog generator.
- Tinkerer - Tinkerer is a blogging engine/.static website generator powered by Sphinx.

进程管理

Libraries for starting and communicating with OS processes.

- envoy - Python subprocess for Humans™.
- sarge - Yet another wrapper for subprocess.
- sh - A full-fledged subprocess replacement for Python.

并发和并行

Libraries for concurrent and parallel execution.

- multiprocessing - (Python standard library) Process-based "threading" interface.

- threading - (Python standard library) Higher-level threading interface.
- eventlet - Asynchronous framework with WSGI support.
- gevent - A coroutine-based Python networking library that uses greenlet.
- Tomorrow - Magic decorator syntax for asynchronous code.

网络

Libraries for networking programming.

- asyncio - (Python standard library) Asynchronous I/O, event loop, coroutines and tasks.
- Tornado - A Web framework and asynchronous networking library.
- Twisted - An event-driven networking engine.
- pulsar - Event-driven concurrent framework for Python.
- diesel - Greenlet-based event I/O Framework for Python.
- pyzmq - A Python wrapper for the ZeroMQ message library.
- txZMQ - Twisted based wrapper for the ZeroMQ message library.

WebSocket处理

Libraries for working with WebSocket.

- AutobahnPython - WebSocket & WAMP for Python on Twisted and asyncio.
- Crossbar - Open-source Unified Application Router (Websocket & WAMP for Python on Autobahn).
- django-socketio - WebSockets for Django.
- WebSocket-for-Python - WebSocket client and server library for Python 2 and 3 as well as PyPy.

WSGI服务器

兼容*WSGI*的*web*服务器.

- gunicorn - Pre-forked, partly written in C.
- uwsgi - A project aims at developing a full stack for building hosting services, written in C.
- bjoern - Asynchronous, very fast and written in C.
- fapws3 - Asynchronous (network side only), written in C.
- meinheld - Asynchronous, partly written in C.
- netius - Asynchronous, very fast.

- paste - Multi-threaded, stable, tried and tested.
- rocket - Multi-threaded.
- waitress - Multi-threaded, poweres Pyramid.
- Werkzeug - A WSGI utility library for Python that powers Flask and can easily be embedded into your own projects.

远程调用

RPC-compatible servers.

- SimpleJSONRPCServer - This library is an implementation of the JSON-RPC specification.
- SimpleXMLRPCServer - (Python standard library) Simple XML-RPC server implementation, single-threaded.
- zeroRPC - zerorpc is a flexible RPC implementation based on ZeroMQ and MessagePack.

加解密

- cryptography - A package designed to expose cryptographic primitives and recipes to Python developers.
- hashids - Implementation of hashids in Python.
- Paramiko - A Python (2.6+, 3.3+) implementation of the SSHv2 protocol, providing both client and server functionality.
- Passlib - Secure password storage/hashing library, very high level.
- PyCrypto - The Python Cryptography Toolkit.
- PyNacl - Python binding to the Networking and Cryptography (NaCl) library.

图形界面（GUI）

Libraries for working with graphical user interface applications.

- curses - Built-in wrapper for ncurses used to create terminal GUI applications.
- enaml - Creating beautiful user-interfaces with Declaratic Syntax like QML.
- kivy - A library for creating NUI applications, running on Windows, Linux, Mac OS X, Android and iOS.
- pyglet - A cross-platform windowing and multimedia library for Python.
- PyQt - Python bindings for the Qt cross-platform application and UI framework, with support for both Qt v4 and Qt v5 frameworks.

- PySide - Python bindings for the Qt cross-platform application and UI framework, supporting the Qt v4 framework.
- Tkinter - Tkinter is Python's de-facto standard GUI package.
- Toga - A Python native, OS native GUI toolkit.
- urwid - A library for creating terminal GUI applications with strong support for widgets, events, rich colors, etc.
- wxPython - A blending of the wxWidgets C++ class library with the Python.
- PyGObject - Python Bindings for GLib/GObject/GIO/GTK+ (GTK+3)
- Flexx - Flexx is a pure Python toolkit for creating GUI's, that uses web technology for its rendering.

游戏开发

Awesome game development libraries.

- Cocos2d - cocos2d is a framework for building 2D games, demos, and other graphical/interactive applications. It is based on pyglet.
- Panda3D - 3D game engine developed by Disney and maintained by Carnegie Mellon's Entertainment Technology Center. Written in C++, completely wrapped in Python.
- Pygame - Pygame is a set of Python modules designed for writing games.
- PyOgre - Python bindings for the Ogre 3D render engine, can be used for games, simulations, anything 3D.
- PyOpenGL - Python ctypes bindings for OpenGL and it's related APIs.
- PySDL2 - A ctypes based wrapper for the SDL2 library.
- PySFML - Python bindings for SFML
- RenPy - A Visual Novel engine.

日志工具

Libraries for generating and working with logs.

- logging - (Python standard library) Logging facility for Python.
- logbook - Logging replacement for Python.
- Eliot - Logging for complex & distributed systems.
- Raven - The Python client for Sentry.
- Sentry - A realtime logging and aggregation server.

测试

Libraries for testing codebases and generating test data.

- 测试框架
  - unittest - (Python standard library) Unit testing framework.
  - nose - nose extends unittest.
  - contexts - A BDD framework for Python 3.3+. Inspired by C#'s `Machine.Specifications`.
  - hypothesis - Hypothesis is an advanced Quickcheck style property based testing library.
  - mamba - The definitive testing tool for Python. Born under the banner of BDD.
  - PyAutoGUI - PyAutoGUI is a cross-platform GUI automation Python module for human beings.
  - pyshould - Should style asserts based on PyHamcrest.
  - pytest - A mature full-featured Python testing tool.
  - pyvows - BDD style testing for Python. Inspired by Vows.js.
  - Robot Framework - A generic test automation framework.
- Web测试
  - Selenium - Python bindings for Selenium WebDriver.
  - locust - Scalable user load testing tool written in Python.
  - sixpack - A language-agnostic A/B Testing framework.
  - splinter - Open source tool for testing web applications.
- Mock
  - mock - (Python standard library) A mocking and patching library.
  - doublex - Powerful test doubles framework for Python.
  - freezegun - Travel through time by mocking the datetime module.
  - httmock - A mocking library for requests for Python 2.6+ and 3.2+.
  - httpretty - HTTP request mock tool for Python.
  - responses - A utility library for mocking out the requests Python library.
  - VCR.py - Record and replay HTTP interactions on your tests.
- 对象工厂
  - factory_boy - A test fixtures replacement for Python.
  - mixer - Another fixtures replacement. Supported Django, Flask, SQLAlchemy, Peewee and etc.
  - model_mommy - Creating random fixtures for testing in Django.
- 代码覆盖率
  - coverage - Code coverage measurement.
- 伪数据（生成）
  - faker - A Python package that generates fake data.
  - fake2db - Fake database generator.
  - radar - Generate random datetime / time.
- 错误处理

    - FuckIt.py - FuckIt.py uses state-of-the-art technology to make sure your Python code runs whether it has any right to or not.

代码分析和优化

Libraries and tools for analysing, parsing and manipulation codebases.

- 代码分析
    - code2flow - Turn your Python and JavaScript code into DOT flowcharts.
    - pycallgraph - A library that visualises the flow (call graph) of your Python application.
    - pysonar2 - A type inferencer and indexer for Python.
- 代码检查和优化
    - Flake8 - The modular source code checker: pep8, pyflakes and co.
    - Pylint - A source code analyzer.
    - pylama - Code audit tool for Python and JavaScript.

调试工具

Libraries for debugging code.

- 调试器
    - ipdb - IPython-enabled pdb.
    - pudb – A full-screen, console-based Python debugger.
    - pyringe - Debugger capable of attaching to and injecting code into Python processes.
    - wdb - An improbable web debugger through WebSockets.
    - winpdb - A Python Debugger with GUI, capable of remote debugging based on `rpdb2`.
    - django-debug-toolbar - Display various debug information for Django.
    - django-devserver - A drop-in replacement for Django's runserver.
    - flask-debugtoolbar - A port of the django-debug-toolbar to flask.
- 性能分析
    - line_profiler - Line-by-line profiling.
    - memory_profiler - Monitor Memory usage of Python code.
    - profiling - An interactive Python profiler.
- 其它
    - pyelftools - Parsing and analyzing ELF files and DWARF debugging information.
    - python-statsd - Python Client for the statsd server.

科学计算和数据分析

Libraries for scientific computing and data analyzing.

- astropy - A community Python library for Astronomy.
- bcbio-nextgen - A toolkit providing best-practice pipelines for fully automated high throughput sequencing analysis.
- bccb - Collection of useful code related to biological analysis.
- Biopython - Biopython is a set of freely available tools for biological computation.
- blaze - NumPy and Pandas interface to Big Data.
- cclib - A library for parsing and interpreting the results of computational chemistry packages.
- NetworkX - A high-productivity software for complex networks.
- Neupy - Running and testing different Artificial Neural Networks algorithms.
- Numba - Python JIT (just in time) complier to LLVM aimed at scientific Python by the developers of Cython and NumPy.
- NumPy - A fundamental package for scientific computing with Python.
- Open Babel - A chemical toolbox designed to speak the many languages of chemical data.
- Open Mining - Business Intelligence (BI) in Python (Pandas web interface)
- orange - Data mining, data visualization, analysis and machine learning through visual programming or Python scripting.
- Pandas - A library providing high-performance, easy-to-use data structures and data analysis tools.
- PyDy - Short for Python Dynamics, used to assist with workflow in the modeling of dynamic motion based around NumPy, SciPy, IPython, and matplotlib.
- PyMC - Markov Chain Monte Carlo sampling toolkit.
- RDKit - Cheminformatics and Machine Learning Software.
- SciPy - A Python-based ecosystem of open-source software for mathematics, science, and engineering.
- statsmodels - Statistical modeling and econometrics in Python.
- SymPy - A Python library for symbolic mathematics.
- zipline - A Pythonic algorithmic trading library.

数据可视化

Libraries for visualizing data. See: awesome-javascript.

- matplotlib - A Python 2D plotting library.
- bokeh - Interactive Web Plotting for Python.

- ggplot - Same API as ggplot2 for R.
- plotly - Collaborative web plotting for Python and matplotlib.
- pygal - A Python SVG Charts Creator.
- pygraphviz - Python interface to Graphviz.
- PyQtGraph - Interactive and realtime 2D/3D/Image plotting and science/engineering widgets.
- SnakeViz - A browser based graphical viewer for the output of Python's cProfile module.
- vincent - A Python to Vega translator.
- VisPy - High-performance scientific visualization based on OpenGL.

视觉计算

Libraries for computer vision.

- OpenCV - Open Source Computer Vision Library.
- SimpleCV - An open source framework for building computer vision applications.

机器学习

Libraries for Machine Learning. See: awesome-machine-learning.

- Crab - A flexible, fast recommender engine.
- gensim - Topic Modelling for Humans.
- hebel - GPU-Accelerated Deep Learning Library in Python.
- NuPIC - Numenta Platform for Intelligent Computing.
- pattern - Web mining module for Python.
- PyBrain - Another Python Machine Learning Library.
- Pylearn2 - A Machine Learning library based on Theano.
- python-recsys - A Python library for implementing a Recommender System.
- scikit-learn - A Python module for machine learning built on top of SciPy.
- vowpal_porpoise - A lightweight Python wrapper for Vowpal Wabbit.

MapReduce

Framworks and libraries for MapReduce.

- dpark - Python clone of Spark, a MapReduce alike framework in Python.
- dumbo - Python module that allows one to easily write and run Hadoop programs.
- luigi - A module that helps you build complex pipelines of batch jobs.

- mrjob - Run MapReduce jobs on Hadoop or Amazon Web Services.
- PySpark - The Spark Python API.
- streamparse - Run Python code against real-time streams of data. Integrates with Apache Storm.

函数式编程

Functional Programming with Python.

- CyToolz - Cython implementation of Toolz: High performance functional utilities.
- fn.py - Functional programming in Python: implementation of missing features to enjoy FP.
- funcy - A fancy and practical functional tools.
- Toolz - A collection of functional utilities for iterators, functions, and dictionaries.

运维开发工具

Software and libraries for DevOps.

- Ansible - A radically simple IT automation platform.
- SaltStack - Infrastructure automation and management system.
- OpenStack - Open source software for building private and public clouds.
- Docker Compose - Fast, isolated development environments using Docker.
- Fabric - A simple, Pythonic tool for remote execution and deployment.
- cuisine - Chef-like functionality for Fabric.
- Fabtools - Tools for writing awesome Fabric files.
- gitapi - Pure-Python API for Git.
- hgapi - Pure-Python API for Mercurial.
- honcho - A Python clone of Foreman, for managing Procfile-based applications.
- pexpect - Controlling interactive programs in a pseudo-terminal like GNU expect.
- psutil - A cross-platform process and system utilities module.
- supervisor - Supervisor process control system for UNIX.

任务调度

Libraries for scheduling jobs.

- APScheduler - A light but powerful in-process task scheduler that lets you schedule functions.

- django-schedule - A calendaring app for Django.
- doit - A task runner and build tool.
- gunnery - Multipurpose task execution tool for distributed systems with web-based interface.
- Joblib - A set of tools to provide lightweight pipelining in Python.
- Plan - Writing crontab file in Python like a charm.
- schedule - Python job scheduling for humans.
- Spiff - A powerful workflow engine implemented in pure Python.
- TaskFlow - A Python library that helps to make task execution easy, consistent and reliable.

使用其它语言扩展Python

Libraries for providing foreign function interface.

- cffi - Foreign Function Interface for Python calling C code.
- ctypes - (Python standard library) Foreign Function Interface for Python calling C code.
- PyCUDA - A Python wrapper for Nvidia's CUDA API.
- SWIG - Simplified Wrapper and Interface Generator.

高性能

Libraries for making Python faster.

- Cython - Optimizing Static Compiler for Python. Uses type mixins to compile Python into C or C++ modules resulting in large performance gains.
- PeachPy - x86-64 assembler embedded in Python. Can be used as inline assembler for Python or as a stand-alone assembler for Windows, Linux, OS X, Native Client and Go.
- PyPy - An implementation of Python in Python. The interpreter uses black magic to make Python very fast without having to add in additional type information.
- Pyston - A Python implementation built using LLVM and modern JIT techniques with the goal of achieving good performance.
- Stackless Python - An enhanced version of the Python.

Windows工具

Python programming on Microsoft Windows.

- Python(x,y) - Scientific-applications-oriented Python Distribution based on Qt and

Spyder.

- pythonlibs - Unofficial Windows binaries for Python extension packages.
- PythonNet - Python Integration with the .NET Common Language Runtime (CLR).
- PyWin32 - Python Extensions for Windows.
- WinPython - Portable development environment for Windows 7/8.

网络虚拟化和SDN

网络虚拟化和*SDN(Software Defined Networking).*

- Mininet - A popular network emulator and API written in Python.
- POX - An open source development platform for Python-based Software Defined Networking (SDN) control applications, such as OpenFlow SDN controllers.
- Pyretic - A member of the Frenetic family of SDN programming languages that provides powerful abstractions over network switches or emulators.
- SDX Platform - SDN based IXP implementation that leverages Mininet, POX and Pyretic.

硬件

Libraries for programming with hardware.

- ino - Command line toolkit for working with Arduino.
- Pyro - Python Robotics.
- PyUserInput - A module for cross-platform control of the mouse and keyboard.
- scapy - A brilliant packet manipulation library.
- wifi - A Python library and command line tool for working with WiFi on Linux.
- Pingo - Pingo provides a uniform API to program devices like the Raspberry Pi, pcDuino, Intel Galileo, etc.

兼容性

Python 2 和 *3*兼容性和转换工具.

- Python-Future - The missing compatibility layer between Python 2 and Python 3.
- Python-Modernize - Modernizes Python code for eventual Python 3 migration.
- Six - Python 2 and 3 compatibility utilities.

开发者插件

各种编辑器和*IDE*的插件.

- Emacs
  - Elpy - Emacs Python Development Environment.
- Sublime Text
  - SublimeJEDI - A Sublime Text plugin to the awesome auto-complete library Jedi.
  - Anaconda - Anaconda turns your Sublime Text 3 in a full featured Python development IDE.
- Vim
  - YouCompleteMe - Includes Jedi-based completion engine for Python.
  - Jedi-vim - Vim bindings for the Jedi auto-completion library for Python.
  - Python-mode - An all in one plugin for turning Vim into a Python IDE.
- Visual Studio
  - PTVS - Python Tools for Visual Studio.

IDEs

Popular Python IDEs.

- PyCharm - Commercial Python IDE by JetBrains. Has free community edition available.
- LiClipse - Free polyglot IDE based on Eclipse. Uses PyDev for Python support.
- Spyder - Open Source Python IDE.

Python资源

Where to discover new Python libraries.

网站

- r/Python
- CoolGithubProjects
- Django Packages
- Full Stack Python
- Python 3 Wall of Superpowers
- Python Hackers
- Python ZEEF
- Trending Python repositories on GitHub today

周报

- Import Python Newsletter
- Pycoder's Weekly
- Python Weekly

Twitter

- @codetengu
- @getpy
- @planetpython
- @pycoders
- @pypi
- @pythontrending
- @PythonWeekly