Contents

[Contents 1](#_Toc147927607)

[IDE使用 4](#_Toc147927608)

[Jupyter Notebook (还有大量的功能，见网上教程) 4](#_Toc147927609)

[PyCharm 4](#_Toc147927610)

[Python debugging in VS Code 6](#_Toc147927611)

[<最佳实践> 10](#_Toc147927612)

[模块与包 10](#_Toc147927613)

[语法(str, list, tuple, set, dict, …,file) 13](#_Toc147927614)

[面向函数编程 18](#_Toc147927615)

[函数 18](#_Toc147927616)

[装饰器 20](#_Toc147927617)

[函数curry化 20](#_Toc147927618)

[filter, map and reduce, sort 21](#_Toc147927619)

[面向对象编程 21](#_Toc147927620)

[类和继承类 21](#_Toc147927621)

[\_\_call\_\_() #可调用对象 22](#_Toc147927622)

[类实现数组功能 23](#_Toc147927623)

[数据类 (python 3.7的新功能) 23](#_Toc147927624)

[技巧 23](#_Toc147927625)

[序列解包sequence unpacking和字典解包map unpacking 23](#_Toc147927626)

[性能优化 24](#_Toc147927627)

[PEP8 –Style Guider for Python Code 25](#_Toc147927628)

[PEP 484 Type Hints (for offline type checker) 27](#_Toc147927629)

[unit test 28](#_Toc147927630)

[Python包 30](#_Toc147927631)

[项目打包及使用包 31](#_Toc147927632)

[Poetry：python项目依赖管理和构建管理 33](#_Toc147927633)

[包管理器pip及虚拟环境 35](#_Toc147927634)

[包管理器Conda及虚拟环境 37](#_Toc147927635)

[Conda-Pack 39](#_Toc147927636)

[Python运行脚本和启动服务 39](#_Toc147927637)

[常用模块 40](#_Toc147927638)

[$import copy, sys, os, glob, tempfile, shutil, base64, logging, smptplib, optparse/argparse, importlib.import\_module 41](#_Toc147927639)

[日期和时间 $import time, datetime, calendar 44](#_Toc147927640)

[数学 $import decimal, fractions, math, random 45](#_Toc147927641)

[$from collections import defaultdict, OrderDict, ChainMap, Counter, deque 46](#_Toc147927642)

[迭代器函数 $from itertools import chain, izip, imap, islice, … 46](#_Toc147927643)

[$import re 46](#_Toc147927644)

[单元测试 48](#_Toc147927645)

[文件解析 48](#_Toc147927646)

[import pprint 48](#_Toc147927647)

[copy and paste 48](#_Toc147927648)

[$import json,csv, np.save/load, pickle.dump/load 48](#_Toc147927649)

[Beautiful Soup --HTML and XML 解析 49](#_Toc147927650)

[PDF,Excel,World解析 49](#_Toc147927651)

[多线程，多进程，定时任务和GUI自动化 52](#_Toc147927652)

[数据库 55](#_Toc147927653)

[Sqlite3 55](#_Toc147927654)

[网络数据采集 56](#_Toc147927655)

[网络请求和响应 --requests 56](#_Toc147927656)

[解析HTML and XML --Beautiful Soup 58](#_Toc147927657)

[模拟浏览器操作 60](#_Toc147927658)

[表单自动填写 61](#_Toc147927659)

[<python基础教程> 61](#_Toc147927660)

[<Python in Practice> 70](#_Toc147927661)

[<Python编程快速上手—让繁琐工作自动化> 74](#_Toc147927662)

[<Effective Python> 75](#_Toc147927663)

[<Flask Web开发> 75](#_Toc147927664)

[tutorial 75](#_Toc147927665)

[视图 76](#_Toc147927666)

[数据库 81](#_Toc147927667)

[电子邮件 82](#_Toc147927668)

[大型程序结构 83](#_Toc147927669)

[OpenAPI via connextion 88](#_Toc147927670)

[PyQt 95](#_Toc147927671)

[UI Component 97](#_Toc147927672)

[styles 101](#_Toc147927673)

[PyQt提供了两种通信机制： 102](#_Toc147927674)

[customized widget 105](#_Toc147927675)

[mix programming: Python <-> C/C++ 109](#_Toc147927676)

[Cython 109](#_Toc147927677)

[Boost.Python 114](#_Toc147927678)

常见问题：

1. Qt.qpa.plugin: could not find the Qt platform plugin “xcb” in “”. This application failed to start because no Qt platform plugin could be initialized. Reinstalling the application may fix this problem.

Import os

os.environ['QT\_QPA\_PLATFORM\_PLUGIN\_PATH'] = r'C:\Users\300230716\Miniconda3\envs\pview\Library\plugins'

1. 若os系统已安装多个python intepreter版本，如何选择默认版本？

$ python –version

$ python3 –version

$ ls /usr/bin/python Tab显示python多个版本

$ sudo update-alternatives –remove python /usr/bin/python2

$ sudo update-alternatives –install /usr/bin/python python /usr/bin/python3.8 10

This gives a priority of 10 for the path of python3.8

$ python –version // python3.8

// link python and python3 to /usr/bin/python3.9

$ sudo alternatives --set python /usr/bin/python3.9

$ sudo alternatives --set python3 /usr/bin/python3.9

# IDE使用

## Jupyter Notebook (还有大量的功能，见网上教程)

<https://www.cnblogs.com/nxld/p/6566380.html>

$sudo pip install jupyter

project$ jupyter notebook #在指定项目下

会打开浏览器

双击\*.ipynb

## PyCharm

安装破解版

<https://blog.csdn.net/u014044812/article/details/78727496>

**远程调试**

<https://blog.csdn.net/ll641058431/article/details/53049453>

远程显示图像

<https://blog.csdn.net/liyachong138/article/details/60141881>

安装python, matplotlib //服务器

安装pycharm专业版 //客户端

通过ssh连接服务器

$ssh -X [user@ip](mailto:user@ip) 然后输入密码

$env 查看DISPLAY (一般=localhost:10.0)

$python

>>>import matplotlib

>>>print matplotlib.get\_backend() (查看输出结果：Qt4Agg)

打开pycharm, 设置**项目的解释器为远程服务器的解释器**

pycharm → file → settings → project interpreter → add remote (点击齿轮）



SSH Credentials

host: ip port: 22 user:qzlin (需要root权限？) password:XXX

python interpreter path: /usr/bin/python (服务器上的python 路径)

test.py

import numpy as np

import matplotlib

matplotlib.use('Qt4Agg')

import matplotlib.pyplot as plt

x = np.linspace(0, 2\*3.14, num=50)

y = np.sin(x)

plt.plot(x, y)

plt.show()

Run → Edit Configurations → Environment variables → add

DISPLAY=localhost.10.0

运行后，客户端会显示matplot图像

**代码模板**

**File -> Settings -> Editor -> File and Code Templates ->**

Python Script:

#!/usr/bin/env python2

# Copyright (c) 2017-present, Tonghuashun, Inc.

"""

-------------------------------------------------

File Name： ${NAME}

Description :

Author : linqizhong@myhexin.coom

date： ${DATE}

-------------------------------------------------

Change Activity:

${DATE}:

-------------------------------------------------

"""

模板变量:

${PROJECT\_NAME} - 当前Project名称;

${NAME} - 在创建文件的对话框中指定的文件名;

${USER} - 当前用户名;

${DATE} - 当前系统日期;

${TIME} - 当前系统时间;

${YEAR} - 年;

${MONTH} - 月;

${DAY} - 日;

${HOUR} - 小时;

${MINUTE} - 分钟；

${PRODUCT\_NAME} - 创建文件的IDE名称;

${MONTH\_NAME\_SHORT} - 英文月份缩写, 如: Jan, Feb, etc;

${MONTH\_NAME\_FULL} - 英文月份全称, 如: January, February, etc；

## Python debugging in VS Code

* **python code/interpreter in the same machine**

<https://code.visualstudio.com/docs/python/python-tutorial#_configure-and-run-the-debugger>

Install vscode

install Extension: Python // turn vs code into python editor

install python interpreter // Linux has built-in Python 3

$ python3 --version

Start VS Code in a workspace folder

$ mkdir hello && cd hello

$ code . // open VS Code (code) in that folder (.)

method 1: **global env (including Python interpreter and Packages, by default)**

Python extension automatically selects the interpreter in the following priority order:

1. Virtual environments located directly under the workspace folder.
2. Virtual environments related to the workspace but stored globally. For example, Pipenv or Poetry environments that are located outside of the workspace folder.
3. Globally installed interpreters. For example, the ones found in /usr/local/bin, C:\\python38, etc.

Create a launch.json -> select a debug configuration: Python File

**launch.json**

"configurations": [

{

"name": "Python: Current File", // appears in the VS Code Run&Debug

"type": "python", // debugger type

**"request": "launch",**

"program": "${file}", // currently active file in the editor

"console": "integratedTerminal", // display output in the debug console

"justMyCode": true //default is true, restricts debugging to user-written code only

}

]

"request": "launch" //start the debugger on the file specified in program

"request": "attach" //attach the debugger to an already running process

"program": "<startup file>" //path to the python program's entry module (startup file)

"python": "<python interpreter>" //default is ${command:python.interpreterPath}

"args": ["--port", "1593"] //arguments of Python program

"stopOnEntry": true //breaks the debugger at the first line of the program being debugged

"console": "integratedTerminal" //"internalConsole", "integratedTerminal", "externalTerminal"

"sudo": true // allows for debugging apps that require elevation

"subProcess": true //Specifies whether to enable subprocess debugging

"cwd": "${workspaceFolder}" //default is ${workspaceFolder}, the folder open in vs code

"justMyCode": true // false enable debugging of standard library functions

"env": // optional env var for the debugger process beyond system env var

"envFile": "${workspaceFolder}/.env" // .env in the current workspace folder

To debug app that requires administrator privileges, use

"console": "externalTerminal"

"sudo": "True".

**main.py**

main.py -> set breakpoint -> Debug Python File

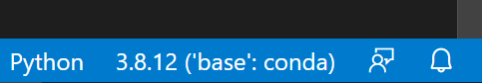
method 2: **local env ( virtual or conda)**

A virtual environment creates a folder (.venv) that contains a copy/symlink to a specific interpreter and packages.

A conda environment is a Python environment that's managed using the conda package manager

1. Create a virtual environment

Ctrl+Shift+P -> Python: Create Environment -> .venv (or .conda) -> create/select the interpreter from computer



// manually specify an interpreter

Ctrl+Shift+P -> Python: Select Interpreter -> Enter interpreter path... e.g.,

.venv/Scripts/python.exe

1. Create a Python source code file

hello

.venv

main.py

1. Configure and run the debugger

main.py -> set breakpoint

click `Run and Debug` -> Python File -> Debug Python File

1. add packages

Command Palette to run Terminal: Create New Terminal (Ctrl+Shift+`). This command opens a command prompt for your selected interpreter.

$ python3 -m pip install numpy

**debugpy：任意机器任意方式启动程序**

* **Command line debugging**

python -m debugpy

--listen | --connect

[<host>:]<port> // host address and port for the debug adapter server

[--wait-for-client] // code run when connection from the debug server

[--configure-<name> <value>]...

[--log-to <path>] [--log-to-stderr]

<filename> | -m <module> | -c <code> | --pid <pid>

[<arg>]...

--pid <pid> //Specifies a process that is already running to inject the debug server into

1. Create a Python source code file

hello

main.py

Create a launch.json -> select a debug configuration: Remote Attach, localhost, 5678

1. launch.json

"configurations": [

{

"name": "Python: Remote Attach",

"type": "python",

**"request": "attach",**

**“connect”: {**

**“host”: “localhost”,**

**“port”: 5678**

**}**

}

]

**host is optional, by default 127.0.0.1**

1. debugpy

Terminal: Create New Terminal, which activates the script's selected environment.

$ python3 -m pip install --upgrade debugpy

$ python3 -m debugpy --listen localhost:5678 --wait-for-client main.py

if python program is launched by other app, insert debugpy inside \*.py directly

\*.py

import debugpy

debugpy.listen(5678)

print("Waiting for client to attach...")

debugpy.wait\_for\_client()

debugpy.breakpoint() // Invoking a breakpoint in code

$ python3 main.py

* **debug python inside docker**

<https://www.python-engineer.com/posts/debug-python-docker/>

hello

main.py

requirements.txt

Dockerfile

requirements.txt

debugpy

Dockerfile

FROM python:3.9

WORKDIR /app

COPY . .

RUN pip install -r requirements.txt

CMD ["python3", "-m", "debugpy", "--listen", "0.0.0.0:5678", "--wait-for-client", "main.py"]

$ docker build . -t hello

$ docker images

$ docker run -p 5678:5678 hello

If you edit code, re-run the above

Create a launch.json -> select a debug configuration: Remote Attach, localhost, 5678

launch.json

"configurations": [

{

"name": "Python: Remote Attach",

"type": "python",

"request": "attach",

“connect”: {

“host”: “localhost”, // could be remote machine

“port”: 5678

},

**"pathMappings": [{**

**“localRoot”: “${workspaceFolder}”, // <project path>**

**“remoteRoot”: “/app” // src code in docker**

**}],**

"justMyCode": true

}

]

main.py -> set breakpoint

click `Run and Debug` -> Python: Remote Attach

if python program is launched by other app, insert debugpy inside \*.py directly

\*.py

import debugpy

debugpy.listen(("0.0.0.0", 5678))

print("Waiting for client to attach...")

debugpy.wait\_for\_client()

Dockerfile

FROM python:3.9

WORKDIR /app

COPY . .

RUN pip install -r requirements.txt

CMD ["launch.sh"] # assume launch other app call \*.py

$ docker build . -t your\_image

$ docker run -p 5678:5678 your\_image

main.py -> set breakpoint

click `Run and Debug` -> Python: Remote Attach

FAQ:

* unable to install a custom module into your Python project.

Add the location to the python.autoComplete.extraPaths setting and restart VS Code.

"python.analysis.extraPaths": [

"~/.local/lib/Google/google\_appengine",

"~/.local/lib/Google/google\_appengine/lib/flask-0.12" ]

# <最佳实践>

## 模块与包

模块放在可以被解释器找到的路径中

import sys, pprint

**pprint.pprint(sys.path) #查看Python解释器的搜索目录**

**help(‘模块’) #查看模块的帮助信息**

help(‘模块.函数’) #查看模块内函数的具体信息

help(对象) #查看对象对应的类的API

dir(‘模块’) #列出模块的类，函数，变量等

$python --version #查看当前使用的python版本

**if \_\_name\_\_ == '\_\_main\_\_': 模块main入口**

包是文件夹

drawing/ #包目录

\_\_init\_\_.py #包代码(drawing模块)

colors.py #包中colors模块

To make all of your functions available when you've imported drawing, you need to put explicit import statements in \_\_init\_\_.py as above

如何引用？

import drawing #import the drawing package

import drawing.colors #import the colors module,使用时需用drawing.colors.fun()

from drawing import shapes #import the shapes model, 使用时只需用shapes.fun()

**推荐项目结构:**

**Recommended Source Code Structure**

project

package1

\_\_init\_\_.py

module1.py

module2.py

package2

\_\_init\_\_.py

module3.py

module4.py

subpackage1

\_\_init\_\_.py

module5.py

在PyCharm IDE里，需要右击主目录project -> Mark Directory as Sources Root

官方推荐采用absolute import，如下：

from package1 import module1

from package1.module2 import function1

from package2 import class1 # 从package2/\_\_init\_\_.py上导入class1

from package2.subpackage1.module5 import function2

包管理器pip

**$apt-get -y install python-pip**

$ pip --help

**$ pip list #列出所有安装包**

pip list will list ALL installed packages (no matter whether they were installed from the Python Package Index or Github, and no matter whether or not they were installed with the pip install command)

**$ pip freeze # list pip packages installed by pip**

$pip install #安装包

$pip install --upgrade #升级包

$pip uninstall #卸载包

$pip show #显示包详细信息

$pip search #搜索包

虚拟环境

**$sudo pip install virtualenv**

$ cd ~/code/myproject/

$ virtualenv env

**$source env/bin/activate 激活虚拟环境**

(env)$ pip install flask 在虚拟环境下工作

**(env)$ deactivate 退出虚拟环境**

(env)$pip freeze > requirements.txt # output installed packages in requirements format

**(env)$pip install -r requirements.txt # 安装清单中的包**

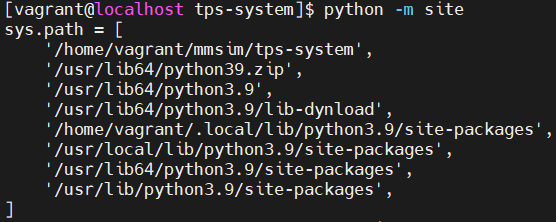
列出安装包

**$ pip list** #列出所有安装包 ，不论是否由pip安装

**$ pip freeze** # list pip packages installed by pip

**$ conda list** # list all packages in a conda environment

$ python -m site # Location of Globally Installed Packages

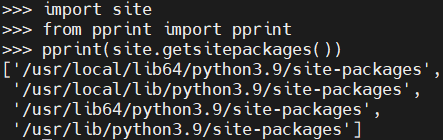


$ python -m site --user-site # location of locally installed packages

$ python

>>> import site

>>> print(site.getsitepackages())



>>> import sys

>>> pprint(sys.path)

A screen shot of a computer

Description automatically generated

* **常用模块**

NLTK(文本处理),

Pandas(数据处理),

sys(系统环境),

os+shutil+tempfile+glob(文件和目录),

base64,

logging,

smptplib,

time+datetime+calendar,

decimal+fractions+math+random,

re,

threading+subprocess,

json+csv+pypdf2+openpyxl+python-docs (文件),

requests+beautifulsoup4

* **模块定义**

A module can define functions, classes and variables.

support.py

def print\_func( par ):

print "Hello : ", par

return

* **模块调用**

# Import module support

import support

# Now you can call defined function that module as follows

support.print\_func("Zara")

The from...import Statement

The from...import \* Statement:

* **模块执行**

python3 -m <module> 解释器会搜索并执行指定的模块，查找模块并运行模块中的\_\_main\_\_函数。

* **模块搜索**

1. **The current directory**.
2. If the module isn't found, Python then searches each directory in the shell variable **PYTHONPATH**.
3. If all else fails, Python checks the default path. On UNIX, **this default path is normally /usr/local/lib/python/.**

## 语法(str, list, tuple, set, dict, …,file)

* **条件测试**

false none 0 “” () [] {} set()被解释为假.

“==”判定同一性(别名)，”is”判定相等性(值相等)

all([True, 1, {3}]) 每个元素都为真时

any([True, 0, []) 至少一个元素为真时

None是NoneType数据类型的唯一值（其他编程语言称为null, nil或undefined

Python 3, 除号的结果默认为浮点数，整除用//

在变量必须保存可变数据类型的值时，例如列表或字典，Python使用引用。对于不可变的数据类型的值，例如字符串、整形或元组，Python变量就保存值本身

|  |  |  |
| --- | --- | --- |
| argument is being passed by reference  the change reflects back in the calling function | | |
| def changeme( mylist ):  mylist.append([1,2,3,4]);  print "Values inside the function: ", mylist  return | mylist = [10,20,30];  changeme( mylist );  print "Values outside the function: ", mylist | Values inside the function: [10, 20, 30, [1, 2, 3, 4]]  Values outside the function: [10, 20, 30, [1, 2, 3, 4]] |
| the reference is being overwritten inside the called function | | |
| def changeme( mylist ):  # This would assig new reference in mylist  mylist = [1,2,3,4];  print "Values inside the function: ", mylist  return | mylist = [10,20,30];  changeme( mylist );  print "Values outside the function: ", mylist | Values inside the function: [1, 2, 3, 4]  Values outside the function: [10, 20, 30] |

* **字符串:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 单引号 | 双引号 | 跨多行-三引号 | 跨多行 | 原始字符串 | unicode字符串 |
| ‘hello world’ | ”hello world” | ’’’hello  Qzlin’’’ | ‘Hello \  Qzlin’ | r’c:\Program Files\fnord’ | u’林其忠’ |

格式化

u'&titles=Category:{0}'.format(category)

方法：find, join/split, lower/upper, replace, strip, translate, endswith, startswith,

’/’.join((‘’, ‘usr’, ‘bin’, ‘env’)) => ‘/usr/bin/env’

import re

regex = re.compile(‘\s+’)

regex.findall(text)

漂亮打印python数据结构

$import pprint

pprint.pprint(sys.path) 用来打印python数据结构

格式化json

json输出 | python -m json.tool

* **列表 （可变，异构，原地修改）**

索引: arr[i]

切片: arr[0:4]

连接(记住：同类型才能相加): [1, 2, 3] + [4]

复制多次: ‘python’\*5, [42]\*10

初始值: []空列表, [0]\*10长度为10，值为0的列表，[none]\*10预留长度为10，没有值的列表

**成员资格: ‘low’ in arr**

**删除元素：arr[i:j] = [] or del arr[i]**

对象的方法: append, count, extend, index, insert, pop, remove, reverse, sort,

for (i, v) in enumerate(arr): #遍历

values[i] = 2\*v ;

for name, age in zip(names, ages): 并行迭代多个序列

…

聚合函数：len, min, max, sum

**列表推导式** （利用其他列表创建新列表，字典或集合）

list1 = [(x, y) for x in range(3) for y in range(3)]

**list2 = [x\*x for x in range(10) if x%3==0]**

**squre\_dict = {x: x\*x for x in range(5)} #{0:0, 1:1, 2:4, 3:9, 4:16}**

**square\_set = {x\*x for x in [1, -1]} #{ 1 }**

**square\_tuple = tuple(x\*x for x in range(5))**

x = [1, 2, 3]

x.extend([4, 5, 6]) == [1, 2, 3, 4, 5, 6]

x.append(7) == [1, 2, 3, 4, 5, 6, 7]

先按第三个元素从大小到排序，然后按第二个元素从小到大排序

x.sort(key=lambda ele: (ele[2], -ele[1]), reverse=True)

* **元组（不可变，常用于函数或方法返回多个值打包成元组）**

def sum\_and\_product(x, y):

return (x+y), (x\*y)

(value\_sum, value\_product) = sum\_and\_product(x, y)

支持操作：in, not in, 比较串联切片和索引, min(), max()

(1, 2, 3) + (4, 5, 6) = (1, 2, 3, 4, 5, 6)

**为了提高程序的可读性，可以给元组的每个元素取一个名字，并通过名字来访问这些元素**

**from collections import namedtuple**

vector = namedtuple(“Dimension”, “x y z”)

voel = vector(1, 2, 3)

voel.x, voel.y, voel.z

* **字典（键不可变，如整形、浮点型、字符串、元组）**

x = {}

x[42] = ‘ff’ #字典可以，自动添加

注意：

x = []

x[42] = ‘ff’ #出错，列表不可以，需要预留空间x = [none] \* 43

方法：clear, copy, fromkeys, get, has\_key, items, iteritems, keys, iterkeys, popitem, setdefault, update,values, itervalues

for (key, value) in scientists.iteritems(): #遍历

…

for k in dict.keys():

for v in dict.values():

for k, v in dict.items():

key in dict #检查字典是否含键

dict.get(key, defaultValue) 获取键值，若不存在则返回初始值

dict.setdefault(key, defaultValue) 检查键，若不存在则设置初始值

**一个defaultdict相当于标准的字典，除了当你查找一个没有包含在内的键时，它用一个你提供的零参数函数建立一个新的键**

**from collections import defaultdict**

word\_counts = defaultdict(int) #int()生成0

word\_counts[word] += 1

dd\_list = defaultdict(list) #list()生成一个空列表

dd\_list[2].append(1) #dd\_list = {2: [1]}

dd\_dcit = defaultdict(dict) #dict()产生一个新字典

dd\_dict["Joel"]["City"] = "Seattle"

#结果：

dd\_list = {

"Joel": {

"City": "Seattle"

}}

**#合并新字典, 若有重要的key,后面覆盖前面的**

merged\_dict = {\*\*dict1, \*\*dict2}

from collections import Counter

counts = Counter([‘American’, ‘China’, …])

counts.most\_common(10) #统计次数，并返回次数最多的前十名

* **集合**

item\_set = set(item\_list)

item in item\_set

cmn\_set = set1.intersection(set2)

unq\_set = set1.union(set2)

* **Types of Operator**

Arithmetic Operators： +， -， \*， 、， %， \*\*， //

Comparison (Relational) Operators： ==, !=, <>, >, <, >=, <=

Assignment Operators： =, +=, -=, \*=, /=, %=, \*\*=, //=

Logical Operators (Bitwise Operators)： &, |, ^, ~, <<, >>

Membership Operators： in, not in

Identity Operators： is, is not

* **生成器和迭代器**

**元组推导得到生成器generator： 迭代按需延迟产生**

SimpleCounter = (x\*\*2 for x in range(10\*\*10)

for val in SimpleCounter:

…

yeild语句创建生成器

def lazy\_range(n):

i = 0

while i < n :

yield i

i += 1

实际范例：

**range(10) 产生[0, ... , 9]列表**

**xrange(10) 产生[0, ..., 9]生成器**

zip(range(10), range(10)　 产生元组列表[(0, 0), (1, 1), …, (9, 9)]

zip(friends, minutes, labels) 产生元组列表[(1, 3, ‘a’), ...]

from itertools import izip 　元组生成器

for val in izip(range(10), range(10):

…

**from itertools import combinations**

a = [1, 2, 3, 4]

aa = combinations(a, 2) #所有(a[i], a[j])的组合，即[(1,2), (1,3), ...(3,4)]

def image\_gen(img\_paths):

for img\_path in img\_paths:

img = imread(img\_path)

yield img

ig = image\_gen(train\_img\_paths)

img0 = next(ig)

img1 = next(ig)

* **文件**

过滤文件

from glob import glob

nii\_files = [f for f in glob(os.path.join(nii\_dir, ‘\*\*/\*.nii.gz’), recursive=True)]

from pathlib import Path

def shallow\_copy(src\_dir, dst\_dir):

for src in glob(os.path.join(src\_dir, ‘\*\*/\*’)):

dst = src.replace(src\_dir, dst\_dir)

os.makedirs(Path(dst).parent, exist\_ok=True)

os.symlink(src, dst, True)

from os import walk, path

files = []

for (dirpath, dirnames, filenames) in walk(folder):

files.extend(filenames)

#文件内容如果可以很小，一次性读取所有内容:

f = open(filename)

for line in f.readlines():

process(line)

f.close()

#文件内容很大，流式处理,一行一行读取

import fileinput

for line in **fileinput.input**(filename):

process(line)

**在python里，一个文件对象就是一个迭代器，每次只处理一行数据，而不是将全部文件加载到内存**

with open("data.txt’") as file:

for line in file:

print len(line.strip())

* **创建进度条**

pip3 install progress

from progress.bar import Bar

bar = Bar(‘Processing’, max=20)

for i in range(20):

…

bar.next()

bar.finish()

**from tqdm import tqdm**

cases = […]

for ele in tqdm(cases):

…

## 面向函数编程

### 函数

**python函数是一等公民，可以赋给变量，或函数参数**

函数可以有默认参数

def my\_print(message="my default message"):

print message

* **函数定义**

def functionname( parameters ):

"function\_docstring"

function\_suite

return [expression]

sum = lambda arg1, arg2: arg1 + arg2;

* **函数传参**

Pass by reference vs value

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example −

def changeme( mylist ):

mylist.append([1,2,3,4]);

print "Values inside the function: ", mylist

return

mylist = [10,20,30];

changeme( mylist );

print "Values outside the function: ", mylist

Values inside the function: [10, 20, 30, [1, 2, 3, 4]]

Values outside the function: [10, 20, 30, [1, 2, 3, 4]]

argument is being passed by reference and the reference is being overwritten inside the called function.

def changeme( mylist ):

mylist = [1,2,3,4]; # This would assig new reference in mylist

print "Values inside the function: ", mylist

return

mylist = [10,20,30];

changeme( mylist );

print "Values outside the function: ", mylist

Values inside the function: [1, 2, 3, 4]

Values outside the function: [10, 20, 30]

* **函数参数**

Required arguments

Keyword arguments：

def printinfo( name, age ):

...

printinfo( age=50, name="miki" )

Default arguments

def printinfo( name, age = 35 ):

...

printinfo( name="miki" )

Variable-length arguments

def printinfo( arg1, \*vartuple ):

...

printinfo( 70, 60, 50 )

结果：arg1=70, vartuple=(60, 50)

An asterisk (\*) is placed before the variable name that holds the values of all nonkeyword variable arguments.

### 装饰器

**使用装饰器改变函数行为　(AOP编程）**

**关键点：函数作为参数，函数内嵌函数，@修饰表示函数执行之前，插入AOP**

import time

**def log\_wrapper(func):**

def wrapper(intext, \*args, \*\*kwargs):

print("logging: {0}({1})".format(func.\_\_name\_\_, intext))

return func(intext)

return wrapper

**@log\_wrapper**

def process(text):

print text

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

process("qzlin")

### 函数curry化

from functools import partial

def foo(a,b,c):

return a + b + c

foo23 = partial(foo, b=23)

foo23(a = 1, c = 3) # => 27

def fun\_wapper(cases, function):

def norm(y):

y\_max, y\_min = np.max(y), np.min(y)

y = (y - y\_min) / (y\_max - y\_min)

return y

y = function(cases)

y = norm(y)

return y

cases = np.array([[val] for val in np.linspace(0, 2\*np.pi, 30)])

partial(fun\_wapper, function=lambda cases: np.sin(cases)+np.cos(cases))(cases)

### filter, map and reduce, sort

filter(function, sequence)

对sequence中的item依次执行function(item)，将执行结果为True的item组成一个List/String/Tuple（取决于sequence的类型）返回

如：filter(lambda x: x!='a', 'abcdef')

map(function, sequence)

对sequence中的item依次执行function(item)，将执行结果组成一个List返回

如：map(lambda x: x+x, 'abcdef')

另外map也支持多个sequence，这就要求function也支持相应数量的参数输入

如：map(lambda x,y: x+y, range(8), range(8))

map(math.pow, [10, 20, 30], [1, 2, 3])

reduce(function, sequence, starting\_value)

对sequence中的item顺序迭代调用function，如果有starting\_value，还可以作为初始值调用

如：reduce(lambda x,y: x+y, range(1,100))

employees = [(‘joe’, 1, 53), (‘qzlin’, 2, 34), … ]

employees\_sorted = sorted(employees, key=lambda emp: emp[0]) 原列表不变

employees.sort(key=lambda emp: emp[0], reverse=True) 原列表改变

## 面向对象编程

### 类和继承类

class Organism(object):

#所有数据通过构造器

def \_\_init\_\_(self, name, x, y):

self.name = name

self.x = x

self.y = y

self.atoms = []

#类似于Java toString()

def \_\_str\_\_(self):

return ‘(%s, [%s, %s])’ % self.name, self.x, self.y)

#公有方法

def move(self):

…

#私有方法

def \_inaccessible(self):

…

#静态方法

@staticmethod

def smeth():

…

python并没有真正的私有化支持，解释器会对上面翻译成

Organism. move(self)

Organism.\_Organism\_\_inaccessible(self)

所以在实际中，

o = Organism()

b.move() #公有方法访问

b.\_Organism\_\_inaccessible() #私有方法访问

# 继承于Organism

class Arthropod(Organism):

def \_\_init\_\_(self, name, x, y, legs):

**super(Arthropod, self).\_\_init\_\_(name, x, y)**

self.legs = legs

#以下子类与父类同名函数，实现多态性

def \_\_str\_\_(self):

…

def move(self):

..

### \_\_call\_\_() #可调用对象

允许一个类的实例像函数一样被调用。实质上说，这意味着 **x() 与 x.\_\_call\_\_() 是相同的**。注意 \_\_call\_\_ 参数可变

应用：\_\_call\_\_ 在那些类的实例经常改变状态的时候会非常有效。调用这个实例是一种改变这个对象状态的直接和优雅的做法。

class Person(object):

def \_\_init\_\_(self, name, gender):

self.name = name

self.gender = gender

def \_\_call\_\_(self, friend):

print 'My name is %s...' % self.name

print 'My friend is %s...' % friend

p = Person('Tom','male')

p('Tony') #实际上调用p.\_\_call\_\_(friend)

callable(p) #判断类实例是否可调用

import torch.utils.data as data

class DatasetFromFolder(data.Dataset):

def \_\_init\_\_(self, image\_dir):

super(DatasetFromFolder, self).\_\_init\_\_()

self.image\_filenames = [join(image\_dir, x) for x in listdir(image\_dir) if is\_image\_file(x)]

### 类实现数组功能

**# 实现索引操作datasets[index]**

def \_\_getitem\_\_(self, index):

input = load\_img(self.image\_filenames[index])

return input

**# 实现len(datasets)**

def \_\_len\_\_(self):

return len(self.image\_filenames)

### 数据类 (python 3.7的新功能)

from dataclasses import dataclass

@dataclass

class Card:

Rank: str

Suit: str

card = Card(“Q”, “heart”)

## 技巧

### 序列解包sequence unpacking和字典解包map unpacking

first, second, \*rest = sequence # 序列解包，可以用来传递任意数量的位置参数

等同于first =sequence[0], second=sequence[1], rest=sequence[2:]

args = (600, 900)

kwargs = dict(copies=2, collate=False)

def print\_args(**\*args**, \*\*kwargs): 序列解包和映射解包

**\*args 是传入位置参数的tuple**

**\*\*kwargs是传入关键字参数的dict**

for arg in args:

…

for key in kwargs:

kwargs[key]

**# derived function or class, skip long input parameters**

def print\_strs(var1, var2, var3):

print(var1, var2, var3)

def print\_strs\_ext(\*\*kwargs):

print\_strs(\*\*kwargs)

print\_strs\_ext(var1=’s’, var2=’t’, var3=’r’)

locals()：囊括了**函数执行到该时间点时所定义的一切变量, 返回的字典**对所有局部变量的名称与值进行映射.常配合mapping unpacking，用来实例化字符串模板

str.format(**\*\*locals()**). #映射解包，可以用来传递任意数量的关键字参数

字符串模板形参需要的实参从locals()返回的dict传入

### 性能优化

$pip install numba

from numba import jit

@jit

def scale\_image(imageData, newRange = (0, 255)):

slice, height, width = len(imageData), len(imageData[0]), len(imageData[0][0])

seq = np.zeros((slice, height, width))

(min, max) = get\_range(imageData)

for z in xrange(slice):

for y in xrange(height):

for x in xrange(width):

val = imageData[z, y, x]

nMin， nMax = newRange[0]， newRange[1]

newVal = (val - min) \* (nMax - nMin) / (max - min) + nMin

seq[z, y, x] = newVal

return seq

ts = time.clock()

traverseImageData(imageData)

print time.clock() - ts, "seconds process time"

$pip install joblib

from joblib import Parallel, delayed

def find\_amicable\_numbers(low, high):

for i in xrange(low, high):

for j in xrange(i + 1, high):

i\_factors\_sum = calc\_primefactors\_sum(i)

j\_factors\_sum = calc\_primefactors\_sum(j)

if i\_factors\_sum == j and i == j\_factors\_sum:

print (i, j)

a = np.arange(1, 10\*\*5+2, 1000)

Parallel(n\_jobs=-1)(delayed(find\_amicable\_numbers)(i, j) for (i, j) in zip(a[:-1], a[1:]))

## PEP8 –Style Guider for Python Code

<https://www.python.org/dev/peps/pep-0008/>

Python 3 disallows mixing the use of tabs and spaces for indentation

with open('/path/to/some/file/you/want/to/read') as file\_1, \

open('/path/to/some/file/being/written', 'w') as file\_2:

file\_2.write(file\_1.read())

# Yes: easy to match operators with operands

income = (gross\_wages

+ taxable\_interest

+ (dividends - qualified\_dividends)

- ira\_deduction

- student\_loan\_interest)

ASCII (in Python 2) or UTF-8 (in Python 3)

Imports should be grouped in the following order:

1. Standard library imports.
2. Related third party imports.
3. Local application/library specific imports.

You should put a blank line between each group of imports.

Absolute imports are recommended

Implicit relative imports should never be used and have been removed in Python 3.

"""This is the example module.

This module does stuff.

"""

from \_\_future\_\_ import barry\_as\_FLUFL

\_\_all\_\_ = ['a', 'b', 'c']

\_\_version\_\_ = '0.1'

\_\_author\_\_ = 'Cardinal Biggles'

import os

import sys

import mypkg.sibling

from mypkg import sibling

from mypkg.sibling import example

Whitespace in Expressions and Statements

Yes: spam(ham[1], {eggs: 2})

Yes: foo = (0,)

Yes: if x == 4: print x, y; x, y = y, x

i = i + 1

submitted += 1

x = x\*2 - 1

hypot2 = x\*x + y\*y

c = (a+b) \* (a-b)

def complex(real, imag=0.0):

return magic(r=real, i=imag)

if foo == 'blah':

do\_blah\_thing()

do\_one()

do\_two()

do\_three()

Write docstrings for all public modules, functions, classes, and methods. Docstrings are not necessary for non-public methods, but you should have a comment that describes what the method does. This comment should appear after the def line.

"""Return a foobang

Optional plotz says to frobnicate the bizbaz first.

"""

公有**no\_leading\_underscore**

Public attributes should have no leading underscores.

For simple public data attributes, it is best to expose just the attribute name, without complicated accessor/mutator methods

私有 **\_single\_leading\_underscore**

weak "internal use" indicator. E.g. from M import \* does not import objects whose names start with an underscore.

私有 **\_\_double\_leading\_underscore**

when naming a class attribute, invokes name mangling (**inside class FooBar, \_\_boo becomes \_FooBar\_\_boo**). Generally, double leading underscores should be used only to avoid name conflicts with attributes in classes designed to be subclassed.

规定函数或变量 **\_\_double\_leading\_and\_trailing\_underscore\_\_**

"magic" objects or attributes that live in user-controlled namespaces. E.g. \_\_init\_\_, \_\_import\_\_ or \_\_file\_\_. Never invent such names; only use them as documented.

避免Python关键词： **single\_trailing\_underscore\_**

used by convention to avoid conflicts with Python keyword, e.g. Tkinter.Toplevel(master, class\_='ClassName'). If your public attribute name collides with a reserved keyword, append a single trailing underscore to your attribute name

Documented interfaces are considered public.

modules should explicitly declare the names in their public API using the \_\_all\_\_ attribute.

explicitly documented part of the containing module's API, such as os.path or a package's \_\_init\_\_ module that exposes functionality from submodules.

All undocumented interfaces should be assumed to be internal.

internal interfaces (packages, modules, classes, functions, attributes or other names) should still be prefixed with a single leading underscore.

Imported names should always be considered an implementation detail

Use ''.startswith() and ''.endswith() instead of string slicing to check for prefixes or suffixes.

Yes: if foo.startswith('bar'):

Object type comparisons should always use isinstance() instead of comparing types directly.

Yes: if isinstance(obj, int):

For sequences, (strings, lists, tuples), use the fact that empty sequences are false.

Yes: if not seq:

if seq:

## PEP 484 Type Hints (for offline type checker)

Python is a dynamically typed language, Type Hints doesn’t cause Python to enforce types, it just suggests types.

* **Any: 任意类型 混用动态类型和静态类型，一般用Any作为应急出口**

a = None # type: Any

a = 2 # ok

s = ‘’ # type: str

s = a # ok

def get(key: str) -> Any:

…

Any 类型对于Python脚本语言的duck typing非常重要，”if it walks like a duck and it quacks like a duck, then it must be a duck”, which means the type of an object is less important than the methods. using duck type you don’t check types at all, instead you check for the presences of a given method or attribute.

class TheHobbit(object):

def \_\_len\_\_(self):

return 100

len(TheHobbit()) == 100

in order to call len(obj), the only real constraint on obj is that it must define a .\_\_len\_\_() method.

* **from typing import Set, List,Tuple, Dict, Sequence, TypeVar**

Basic types: str, float, int, bool

Composite types: List[str], Tuple[int, int], Dict[str, bool]

var : Optional[T] # 变量var可以是类型T or None

List[element\_type]

Dict[key\_type, value\_type]

Set[element\_type]

Tuple(int, int, int)

Callable[[Arg1Type, Arg2Type], ReturnType]

Card = Tuple[str, str] Type Aliases

Deck = List[Card]

def greeting(name: str = ‘World’) -> str:

return ‘Hello ‘ + name

def retry(url: str, retry\_count: int) -> None:

…

def feeder(next\_item: Callable[[], str]) -> None:

next\_item()

* **T = TypeVar(‘T’) # can be anything**

T = TypeVar(‘T’, str, bytes) # must be str or bytes

def first(seq: Sequence[T]) -> T :

return seq[0]

## unit test: TestCase, mock, coverage

**from unittest import TestCase, TestSuite, TestRunner, TestFixture**

TestFixture测试固件：测试准备工作，如创建临时数据库，文件和目录等，方法：setUp()，setDown()。举例：测试百度搜索selenium这个场景， setUp()里写打开浏览器，浏览器最大化，和打开百度首页等脚本代码；tearDown()里写结束搜索后，退出并关闭浏览器的代码。

test case测试用例：test开头的函数。unittest会自动化识别test开头的函数是测试代码，若函数不是test开头，unittest不会执行。

test suite测试套件：测试用例的集合。

test runner（测试执行器 ）：test runner是一个用来执行加载测试用例，并执行用例，且提供测试输出的一个组建。test runner可以加载test case或者test suite进行执行测试任务。

整个的流程就是首先要写好TestCase，然后由TestLoader加载TestCase到TestSuite，然后由TestTestRunner来运行TestSuite，运行的结果保存在TextTestReusult中，整个过程集成在unittest.main模块中。

from selenium import webdriver

import unittest

class BaiduSearch(unittest.TestCase):

　　def setUp(self): # 测试准备工作

　　　　self.driver = webdriver.Chrome()

　　　　self.driver.maximize\_window()

　　　　self.driver.implicitly\_wait(8)

　　　　self.driver.get("https://www.baidu.com")

　　def test\_baidu\_search(self): # test case

　　　　self.driver.find\_element\_by\_id('kw').send\_keys('selenium')

　　　　time.sleep(1)

　　　　assert 'selenium' in self.driver.title

　　def tearDown(self): # 测试结束后的操作，这里基本上都是关闭浏览器

　　　　self.driver.quit()

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

　　unittest.main()

**from unittest.mock import patch, MagicMock**

函数功能需要联网，需要设备等，如何测试？

Mock模块就是在单元测试中模拟部分代码的模块，比如某个函数需要调用其他函数，这个时候我们可以模拟这个第三方函数的结果来略过实际调用它，不光可以节省时间，也可以避免因为第三方函数出错而影响自己的代码，甚至可以很轻松的模拟难以出现的各种情况。

Mock就是模拟某些模块的行为，比如你有一个函数调用了另一个函数，而另一个函数的代码本身不是你写的，或者不需要在当前单元测试中测试，你只是希望拿到另一个函数返回的结果，这个时候就可以用mock来模拟那个函数来略过各种中间过程而直接得到结果。

**若调用的外部代码是面向过程，用mock.patch**

linux\_tool.py

def check\_cmd\_response():

response = send\_shell\_cmd() # 调用第三方函数

…

test\_linux\_tool.py

from unittest import TestCase, mock

class TestLinuxTool(TestCase):

@mock.patch(“linux\_tool.send\_shell\_cmd”)

def test\_check\_cmd\_response(self, mock\_send\_shell\_cmd):

mock\_send\_shell\_cmd.return\_value = “Response from …” #模拟返回

mock\_send\_shell\_cmd.side\_effect = Exception(“Raise Exception”) # 模拟异常

status = linux\_tool.check\_cmd\_response()

self.assertTrue(status)

$ pytest -v –html=~/public\_html/report.html/report.html test\_inux\_tool.py

**如果是面向对象的方法，用mock.patch.object**

linux\_tool.py

class LinuxTool(object):

def check\_cmd\_response(self):

response = send\_shell\_cmd() # 调用第三方函数

…

test\_linux\_tool.py

from unittest import TestCase, mock

class TestLinuxTool(TestCase):

@mock.patch.object(LinuxTool, “send\_shell\_cmd”)

def test\_check\_cmd\_response(self, mock\_send\_shell\_cmd):

mock\_send\_shell\_cmd.return\_value = “Response from …”

status = linux\_tool.check\_cmd\_response()

self.assertTrue(status)

MagicMock模拟第三方类

mock\_object =MagicMock()

mock\_object.method\_A.return\_value = 10

mock\_object.method\_B.return\_value = 20

统计Python代码覆盖率：Coverage

分支覆盖率统计，生成 HTML 格式的统计报告，而且可以集成到 Jenkins 中使用

$ pip3 install coverage

**main.py**

def get\_level(course):

if course >= 90:

return "excellent"

elif course >= 80:

return "great"

elif course >= 60:

return "good"

elif course >= 40:

return "bad"

else:

return "very bad"

# unit test case

**get\_level\_test.py**

import unittest

from main import \*

class GetLevelTest(unittest.TestCase):

def test\_get\_level1(self):

self.assertEquals(get\_level(90), "excellent")

def test\_get\_level2(self):

self.assertEquals(get\_level(80), "great")

if \_\_name\_\_ == "\_\_main\_\_":

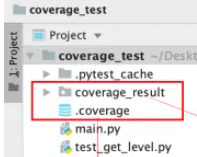
unittest.main(verbosity=2)

* **Coverage命令行**

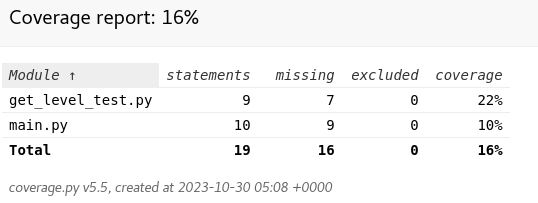
$ coverage help

$ coverage run get\_level\_test.py # 搜集被测代码覆盖率信息，保存到.coverage文件

$ coverage html -d coverage\_result # 生成覆盖率统计结果报告



$ firefox coverage\_result/index.html & # 查看test report



* **Coverage API**

**exec\_api.py**

import coverage

import unittest

cov = coverage.coverage()

cov.start()

suite = unittest.defaultTestLoader.discover("./", "get\_level\_test.py")

unittest.TextTestRunner().run(suite)

cov.stop()

cov.save()

cov.report() # 命令行模式展示结果

cov.html\_report(directory='result\_html') # 生成HTML 覆盖率报告

# Python包

在 Python 开发环境中最常见任务是 Python 版本管理、虚拟环境、包管理、打包发布、安装。

venv 是 Python 标准库支持的虚拟环境创建库, 在 Python 3.5 后推荐使用, 只能创建与主环境 Python 版本一致的虚拟环境。

virtualenv 是 Python 第三方库，可以创建独立的虚拟环境, 支持 Python2 和 Python3.

conda 支持下载创建不同版本 Python 虚拟环境。

poetry 并不提供虚拟环境管理相关功能，只使用虚拟环境, poetry 定位是在单项目全面管理

Python 包的分发可以分为两种：

1. 以源码包的方式发布： 源码包安装的过程，是先解压，再编译，最后才安装，所以它是跨平台的，由于每次安装都要进行编译，相对二进包安装方式来说安装速度较慢。常见的格式：.zip, .tar.gz, .tar.gz2, .tar.Z, .tar
2. 以二进制包形式发布: 二进制包的安装过程省去了编译的过程，直接进行解压安装，所以安装速度较源码包来说更快.由于不同平台的编译出来的包无法通用，所以在发布时，需事先编译好多个平台的包. 常见的格式：.egg, .whl Egg 格式是由 setuptools 在 2004 年引入，而 Wheel 格式是由 PEP427 在 2012 年定义。Wheel 的出现是为了替代 Egg

## 项目打包及使用包

1. 项目打包（传统方法setup.py和现代方法pyproject.toml）
2. 上传软件包到 PyPI $ twine upload dist/dbm-agent-8.31.1.tar.gz
3. 下载安装包并安装 $ pip install dbm-agent
4. 在业务代码中引入第三方软件包 import dbma

**现代方法pyproject.toml**

Folder Structure (官方推荐)

├── LICENSE

├── README.md

├── pyproject.toml

├── src

│ └── npts # src 下面是包名，包下面是业务代码

│ ├── \_\_init\_\_.py

│ └── core.py

└── tests

pyproject.toml

[project]

name = "npts"

version = "0.0.1"

[build-system]

requires = ["hatchling"]

build-backend = "hatchling.build"

**安装 build 依赖并用 build 来打包**

$ python3 -m pip install --upgrade build # 安装依赖

$ python3 -m build # 打包

**把打包好的软件包上传到 PyP**

$ twine upload dist/npts-0.0.1-py3-none-any.whl // requires username/password

Uploading distributions to https://upload.pypi.org/legacy/

下载安装软件包

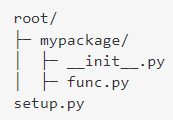
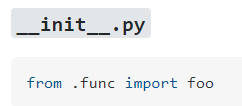
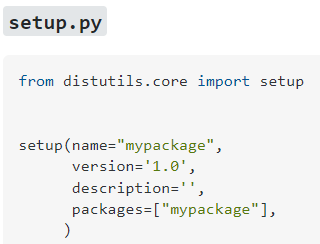
$ pip3 install npts

from npts import core

* 传统方法setup.py

<https://blog.konghy.cn/2018/04/29/setup-dot-py/>

Folder Structure

**打包：将源代码封装，使用者即装即用**

# setup.py

from setuptools import setup, find\_packages

setup (

name = "RedisRun", #包名称

version = "1.0" , #包版本

description = "simple description",

long\_description = "description in detail",

license = 'GPLv3",

platforms = "any", #程序适用的软件平台列表

author = "Qizhong Lin", #程序作者

author\_email = "qizhong.lin@philips.com",

#需要打包的目录列表

# package = find\_package(exclude=["\*.tests", ...]), # 搜索和setup.py同一目录下各个含有\_\_init\_\_.py的包, 排除一些特定的包

package = ['DrQueue'],

# 需要安装的依赖

install\_requires = [

"numpy",

"scipy",

"redis>=2.10.5'

],

# 在Python/scripts下生成exe文件

entry\_points = {'console\_scripts': [

‘redis\_run = DrQueue.RedisRun.redis\_run:main',

]}

)

打包及包安装

Build and Install **Activate your project environment beforehand**

(venv) $ python setup.py sdist # 生成.tar.gz

(venv) $ pip install ./dist/mypackage-1.0.tar.gz

(venv) $ python setup.py bdist\_wheel # 生成whl包

(venv) $ pip wheel –wheel-dir=/local/wheels pkg # 离线安装whl包

(venv) $ pip install 模块名 # 在线安装, 安装该包的相关依赖包

(venv) $ python setup.py install # 下载源码包然后在本地安装，不会安装该包的相关依赖包

Installs the package in whichever environment is active. If you want to install it in a virtualenv, then you need to activate it first. otherwise it will install globally

范列：

$ git clone <https://gitlab.pfh.research.philips.com/tron/trauma>

下载pre-build omnilearn, Set trauma python interpreter -> ominilearn/bin/python3

下载Omnilearn开发版

$ git clone <https://github.com/philips-internal/omnilearn.git>

(pre-built omnilearn) $ python setup.py install 更新omnilearn 代码

## Poetry：python项目依赖管理和构建管理

poetry 将所有的配置都放置在一个 toml 文件中，这些配置包括：**依赖管理、构建、打包、发布**。

灵感来自composer (PHP) 和 cargo (Rust) 。

Poetry支持自动管理Python项目的依赖关系，可以通过简单的命令添加或删除依赖项。

Poetry可以帮助开发者**将Python项目打包成Wheel或sdist格式**，并支持通过pip安装或上传到PyPI等包仓库。

Poetry支持Windows、MacOS和Linux等多个平台，可以在不同操作系统上运行和管理Python项目。

// install poetry (in $HOME/.local/bin on Unix.)

<https://python-poetry.org/docs/>

$ python3 --version // make sure version > 3.7

$ curl -sSL https://install.python-poetry.org | python3 –

$ poetry –version

$ poetry self update // update poetry

$ poetry self update 1.2.0 // update to specific version

$ curl -sSL https://install.python-poetry.org | python3 - --uninstall // uninstall poetry

$ poetry build

$ poetry publish

$ poetry env list //列出所有虚拟环境

$ poetry env use python3 //创建虚拟环境，使用与安装 Poetry 相同的 Python 版本

$ **poetry install**  //检查pyproject.toml文件中依赖项，解析并安装它们

$ **poetry add** requests //将最新版本的requests库添加到您的项目中

$ poetry add black -D //添加开发依赖项black

**rp\_poetry/pyproject.toml**

[tool.poetry.dependencies]

python = "^3.9"

requests = "^2.26.0"

[tool.poetry.dev-dependencies]

pytest = "^5.2"

black = "^21.9b0"

当您运行该poetry add命令时，Poetry 会自动更新pyproject.toml并固定poetry.lock文件中的已解析版本。通过运行poetry lock，Poetry 处理pyproject.toml文件中的所有依赖项并将它们锁定到poetry.lock文件中，使用poetry.lock，您可以确保您使用的版本与其他开发人员使用的版本完全相同

$ poetry lock //更新引脚依赖关系文件

$ poetry lock --no-update //仅解析新的依赖项,不更改文件现有依赖项

$ **poetry install** //读取poetry.lock文件并安装其中声明的所有依赖项

$ poetry show --help --tree //详细地以树形式列出依赖项

将poetry添加到现有项目

$ poetry init //交互式会话以创建pyproject.toml文件

$ **poetry run python3** main.py //在poetry虚拟环境中运行

$ poetry add `cat requirements.txt` // 添加requirements.txt包

$ poetry export --output requirements.txt //poetry项目导出requirements.txt



## 包管理器pip及虚拟环境

**pip类似RedHat的yum, Ubuntu的apt-get, Ruby的gem**

PyPI is a repository for open-source third-party Python packages. It's similar to RubyGems in the Ruby world, PHP's Packagist, CPAN for Perl, and NPM for Node.js.

Install easy\_install

$ sudo apt-get install python-setuptools python-dev build-essential

$ sudo easy\_install pip

or

**$apt-get -y install python-pip**

Python has primitive package manager called easy\_install, which is installed automatically when you install Python itself

$pip --help

$pip list 列出已安装包

**$pip install 安装包**

$pip install --upgrade 升级包

**$pip uninstall 卸载包**

$pip show 显示包详细信息

$pip search 搜索包，类似yum里的search

注：若需要代理$pip install **--proxy=http://185.46.212.34:10015/** numpy

Offline 安装包

1. 若是virtualbox,主机和虚拟机可以共享文件夹http://blog.csdn.net/longerzone/article/details/32119457

virtualbox -> 共享文件夹 -> 共享文件夹路径(E:/pypi) + 共享文件夹名称(pypi)

在虚拟机里：映射共享文件夹pypi -> Ubuntu /home/uftp

$sudo /sbin/mount.vboxsf pypi /home/uftp

$ cd /home/uftp 工作在/home/uftp目录下，即在E:/pypi目录下

下载python package于local repository E:/pypi

$ pip install **--proxy=http://185.46.212.34:10015/** --download . scipy

Copy E:/pypi to Ubuntu /home/qzlin/pypi

**离线安装python package**

**$ pip install --no-index --find-links=file:/home/pypi scipy**

**离线安装包**

**(omnilearn)$ pip install xxx.tar.gz 进入虚拟环境**

* **virtual environment**

virtualenv is a tool for creating isolated Python environments containing their own copy of python, pip, and their own place to keep libraries installed from PyPI.

simply a directory that contains a complete copy of everything needed to run a Python program, including a copy of the python binary itself, a copy of the entire Python standard library, a copy of the pip installer, and (crucially) a copy of the site-packages directory mentioned above.

$sudo **pip install virtualenv**

若有代理：

$export https\_proxy=http://web-proxy.mydomain.com

$sudo -E pip install virtualenv

Usually pip and virtualenv are the only two packages you ever need to install globally.In fact, virtualenv comes with a copy of pip which gets copied into every new environment you create, so virtualenv is really all you need.

$ cd ~/code/myproject/

$ **virtualenv env**

Start by changing directory into the root of your project directory, and then use the virtualenv command-line tool to create a new environment

It's a common convention to call this directory env, and to put it inside your project directory (so, say you keep your code at ~/code/projectname/, the environment will be at ~/code/projectname/env/ - each project gets its own env).

Note: if you're using a version control system like git, you shouldn't commit the env directory. Add it to your .gitignore file (or similar).

projectname

env

bin

include

lib

helloworld.py

helloworld.py

import requests

print requests.get('http://www.baidu.com')

$env/bin/pip install requests

$env/bin/python helloworld.py

可以简约如下：

$**source env/bin/activate** 激活虚拟环境

虚拟环境Python解释器的路径就被添加进PATH中，但这种改变不是永久性的，它只会影响当前的命令行会话

**(env)$** pip install flask

(env)$ python

>>> ...正常python命令行操作

(env)$ **deactivate** 退出虚拟环境

virtualenv默认是采用python2解析器，所以以上生成python2环境

若要生成python3环境呢？

1. pip install --upgrade virtualenv
2. virtual -p python3 env 本地目录下生成env, 即python3环境
3. source env/bin/activate
4. (env)$pip install -r requirements.txt

* **Requirements files**

Requirements files give you a way to create an environment: a set of packages that work together.

requirements make explicit, repeatable installation of packages.

requirements.txt

MyApp

Framework==0.9.4

Library>=0.2

$pip install -r requirements.txt

Output installed packages in requirements format

$pip freeze > requirements.txt

## 包管理器Conda及虚拟环境

安装Anaconda时自带conda

.condarc （需要第一次使用conda config 创建）

ssl\_verify: False

proxy\_servers: #代理设置

http: **http://185.46.212.34:10015**

https: **http://185.46.212.34:10015**

channels: #设置国内镜像

- <https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/>

- <https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge/>

- <https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/msys2/>

#设置国内镜像

$ conda config --add channels <https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/>

注意若conda 因为proxy问题，检查https: [**http://185.46.212.34:10015**](http://185.46.212.34:10015) **or** ssl\_verify: False

$conda –V #检验是否安装以及当前conda的版本

$conda update conda #检查更新当前conda

**$conda env list** 或 conda info -e #查看当前存在哪些虚拟环境,

by default , Env location is in conda/envs, meanwhile, base is default env

# 创建Python虚拟环境

anaconda 命令创建python版本为X.X、名字为your\_env\_name的虚拟环境。your\_env\_name文件可以在Anaconda安装目录envs文件下找到。

conda\_packagelist.txt 是要安装在环境中的包的列表

$**conda create -n your\_env\_name** **python=X.X**（2.7、3.6等） #其它包可以进虚拟环境再装

$conda create --name your\_env\_name --file conda\_packagelist.txt#包文件列表，在虚拟环境生成

**$ conda env create –f environment.yaml** #通过环境文件创建环境，但是pip包还是得另行安装

# 使用激活(或切换不同python版本)的虚拟环境

$ **conda activate your\_env\_name**

(your\_env\_name) $ conda list #查看安装了哪些包。

# 对虚拟环境中安装额外的包

(your\_env\_name)$ **conda install** package #安装package到your\_env\_name

(your\_env\_name)$ pip install …

(your\_env\_name)$ conda install --file requirements.txt

# 删除环境中的某个包。

(your\_env\_name)$ conda remove package\_name

# 导出虚拟环境安装包

(your\_env\_name)$ conda list –explicit > **conda\_packagelist.txt**

**$ conda env export > environment.yaml** #环境文件，在虚拟环境生成

# 关闭虚拟环境(即从当前环境退出返回使用PATH环境中的默认python版本)

$**conda deactivate**

# 删除虚拟环境。

$ conda remove -n your\_env\_name(虚拟环境名称) --all

若虚拟环境，混合安装如conda and pip, 只能导出environment.yaml

## Conda-Pack

<https://conda.github.io/conda-pack/>

One common use case is packing an environment on one machine to distribute to other machines that may not have conda/python installed

On the source machine

(my\_env) qzlin $ # packgage the current environment

(my\_env) qzlin $ conda pack # Pack environment my\_env into my\_env.tar.gz

On the target machine

# Unpack environment into directory `my\_env`

$ mkdir -p my\_env

$ tar -xzf my\_env.tar.gz -C my\_env

# Activate the environment. This adds `my\_env/bin` to your path

$ source my\_env/bin/activate

# Run python from in the environment

(my\_env) $ python

In case you have Conda already installed in your path, you can then add the created environment to the list of environments using:

$ conda config --append envs\_dirs path/to/envs

$ conda activate omnilearn

## Python运行脚本和启动服务

* **Interactive Mode Programming**

$ python

>>> print "Hello, Python!"

>>> exit()

* **Script Mode Programming**

test.py

print "Hello, Python"

$ python test.py

* **like shell**

**#!/usr/bin/python**

print "Hello, Python!"

**$ chmod +x** test.py # This is to make file executable

$./test.py

* **build-in web server to setup a web server**

$python -m SimpleHTTPServer 8080 默认是8000

$curl localhost:8080

相当于ls .

$curl localhost:8080/static/data.txt

相当于cat static/data.txt

* **输入输出**

**STDOUT**

#!/usr/bin/python

print """Hello, Python!

I am qzlin,

please welcome me"""

$./helloworld.py | grep py

**STDIN**

#!/usr/bin/python

import sys

for line in sys.stdin:

print line

$./helloworld.py | ./stream-read.py

**脚本输入参数**

import sys

print 'Then command line arguments are: '

for (i, v) in enumerate(sys.argv):

print i, "\t", v

$./argsv-ex.py arg1 arg2 arg3

1. ./argv-ex.py
2. arg1
3. arg2
4. arg3

推荐用import argparse

## 常用模块

高级文本处理NLTK程序包

大量数值型数据，Pandas程序包

### $import copy, sys, os, glob, tempfile, shutil, base64, logging, smptplib, optparse/argparse, importlib.import\_module

**$import copy**

copy.copy()复制列表或字典

copy.deepcopy()复制列表 （含嵌套列表）

**$import sys**

处理Python运行时配置以及资源，从而可以与前当程序之外的系统环境交互

sys.stdin 标准输入流；sys.stdout 标准输出流；sys.stderr 标准错误流。

sys.path 查找模块所在目录的目录名列表。

sys.argv 命令行的参数列表，第一项脚本名称，第二项第一个命令行参数

sys.platform 返回当前系统平台，如：win32、Linux等。

$import os

Python的系统编程的操作模块，可以处理文件和目录

os.sep 更改操作系统中的路径分隔符。

os.getcwd()获取当前路径，这个在Python代码中比较常用。

os.listdir() 列出当前目录下的所有文件和文件夹。

os.remove() 方法可以删除指定的文件。

os.system() 方法用来运行shell命令。

os.chdir() 改变当前目录，到指定目录中。

os.makedirs() 创建新文件夹

**$import os.path**

os.path：多平台上处理文件

os.path.split('/one/two/three') 解析路径

parts = ('/', 'one', 'two', 'three')

os.path.join(\*parts) /one/two/three

os.path.isfile(file)

os.path.isdir(file)

os.path.exists(file)

**import glob**

glob.glob('dir/\*') 匹配目录dir中的所有路径名（文件或目录）

glob.glob('dir/file?.txt')

glob.glob('dir/\*[0-9].\*') 字符区间

**import tempfile**

tempfile模块提供了多个函数来安全地创建临时文件系统资源

临时文件系统对象，文件关闭时会自动删除

TemporaryFile() 打开并返回一个未命名的文件

temp = tempfile.TemporaryFile()

NamedTemporaryFile()打开并返回一个命名文件

with tempfile.NamedTemporaryFile() as temp:

mkdtemp()创建一个临时目录，并返回其目录名

dir\_name = tempfile.mkdtemp()

tempfile.gettempdir() 获取临时文件位置

查找过程：

环境变量TMPDIR -> 环境变量TEMP -> 环境变量TMP -> window(C:\TMP, ...), Linux(/tmp, /var/tmp, usr/tmp) -> 当前工作目录

高级文件操作

**import shutil**

shutil.copyfile('main.py', 'main.py.copy')

shutil.copy('main.py', 'script') 类似于unix的cp,若目标是目录，则copy文件于该目录下

shutil.move('main.py', 'main.txt') 类似于unix的mv,改名或移动文件

shutil.copytree(src, dst) 复制整个文件夹

shutil.rmtree(path) 删除整个文件夹

import send2trash

send2trash.send2trash('bacon.txt') 删除至回收站

**$import base64**

info = “qizhonglin”

info\_b64encode = base64.b64encode(info)

info\_copy = base64.b64decode(info\_b64encode )

**$import logging**

日志信息保存到一个自己设定格式的文件中

loggin.basicConfig(level=loggin.DEBUG, format=' %(asctime)s - %(levelname)s - %(message)s')

#logging.diable(logging.CRITICAL) 禁用接下来的日志

logging.debug('Start of program')

**$import smptplib**

**$import email.utils**

$from email.mime.text import MIMEText

SMTP（简单邮件传输协议）规定电子邮件应该如何格式化、加密、在邮件服务器之间传递，以及在你点击发送后，计算机要处理的所有其他细节

处理Email，发送、接收、抄送、下载邮件内容等操作

msg = MIMEText('这是消息的主体部分.')

msg['To'] = email.utils.formataddr(('admin', 'admin@example.com'))

msg['From'] = email.utils.formataddr(('Author', 'author@example.com'))

msg['Subject'] = 'Simple test message'

smtpObj = smtplib.SMTP(‘smtp.gmail.com’, 587) 连接到SMTP服务器

try:

smtpObj.ehlo() 向SMTP电子邮件服务器打招呼

smtpObj.starttls() 开始TLS加密

smtpObj.login(‘author@example.com’, ‘password’) 登录到SMTP服务器

smtpObj.sendmail('author@example.com', ['admin@example.com'], msg.as\_string())

finally:

smtpObj.quit()

IMAP（因特网消息访问协议）规定了如何与电子邮件服务提供商的服务器通信，取回发送到你的电子邮件地址的电子邮件

//用IMAP获取和删除电子邮件

import imapclient, pyzmail

imapObj = imapclient.IMAPClient('imap.gmail.com', ssl=True) 连接IMAP服务器

imapObj.login('author@gmail.com', 'password') 登录

imapObj.select\_folder('INBOX', readonly=True) 搜索电子邮件

UIDs = imapObj.search(['SINCE 05-Jul-2014']) 执行搜索

rawMessages = imapObj.fetch([UIds], ['BODY[]', 'FLAGS']) 取邮件

//解析邮件内容

message = pyzmail.PyzMessage.factory(rawMessages[UIDs[i]['BODY[]'])

message.get\_subject()

message.get\_addresses('from')

message.get\_addresses('to')

message.get\_addresses('cc')

message.html\_part.get\_payload().decode(message.html\_part.charset)

参数命令行化

parser = optparse.OptionParser()

parser.add\_option('-u', '--url',

dest="url",

default="http://161.92.142.80:8080/mediawiki")

options, remainder = parser.parse\_args()

url = options.url

python main.py -u http://..."

**import argparse**

def tuple\_type(strings):

strings = strings.replace("(", "").replace(")", "")

mapped\_int = map(int, strings.split(","))

return tuple(mapped\_int)

parser = argparse.ArgumentParser()

parser.add\_argument('integer', type=int, help='display an integer')

parser.add\_argument("--spacing", type=tuple\_type, default=’(0.5, 0.5, 1.0)’)

args = parser.parse\_args()

args.integer

add\_argument(name or flags...[, action][, nargs][, const][, default][, type][, choices][, required][, help][, metavar][, dest])

name or flags - 选项字符串的名字或者列表，例如 foo 或者 -f, --foo。

default - 不指定参数时的默认值。

type - 命令行参数应该被转换成的类型。

help - 参数的帮助信息，当指定为 argparse.SUPPRESS 时表示不显示该参数的帮助信息.

dest - 解析后的参数名称，默认情况下，对于可选参数选取最长的名称，中划线转换为下划线.

**$import importlib.import\_module**

├── clazz

│ ├── \_\_init\_\_.py

│ ├── a.py

│ └── b.py

└── main.py

main.py

import importlib

a = importlib.import\_module(“clazz.a”)

a.show()

### 日期和时间 $import time, datetime, calendar

$import time

time.time() 返回从纪元以来的秒数

time.ctime() 返回可读的时间

time.clock() 返回处理器时钟时间，用于性能测试、基准测试等

time.gmtime() 以UTC格式返回当前时间

time.localtime() 返回应用了当前时区的当前时间

time.sleep(5) 程序暂停5秒，阻塞直到5秒时间过去，不要一次调用time.sleep(30)来暂停30秒，而是使用for循环执行30次time.sleep(1)调用

for i in range(30):

time.sleep(1)

解析和格式化时间

now = time.ctime()

parsed = time.strptime(now)

print ' tm\_year : ', parsed.tm\_year

time.strftime("%a %b ...", parsed)

$import datetime

t = datetime.time() 时间

t.hour, t.minute, ...

today = datetime.date.today() 日期

today.tm\_year, today.tu\_mon, today.tm\_hour, ...

时间运算

yesterday = today - datetime.timedelta(days=1)

tommorrow = toda + datetime.timedelta(days=1)

today > yesterday

today < tommorrow

$import calendar

c = calendar.TextCalendar(calendar.SUNDAY)

c.prmonth(2011, 7) 返回一个月的格式化文本输出

print c.formatyear(2011, 2, 1, 1, 3) 返回一年的格式化文本输出

### 数学 $import decimal, fractions, math, random

$import decimal

Decimal类实现定点和浮点运算，构造函数取一个整数或字符串作为参数,重载了算术运算符

decimal.Decimal('3.14') + decimal.Decimal(str(f))

$import fractions

Fraction类基于numbers模块中Rational定义的API，实现了有理数的数值运算,重载了算术运算符

fractions.Fraction(1, 2) ==1/2

fractions.Fraction('1/2')

fractions.Fraction('0.5')

$import math

pring "pi = %.30f" % math.pi 取小数点后30位的值

math.trunc()

math.floor()

math.ceil()

math.modf() 返回tuple,输入值的小数和整数

math.fabs()

math.pow(x, y)

math.log()

math.exp()

math.radians() <-> math.degrees()

math.hypot(deltaX, deltaY)

$import random

生成随机浮点数、整数、字符串，甚至帮助你随机选择列表序列中的一个元素，打乱一组数据等。

random.seed(1)

random() 返回0<=n<1之间的随机实数n；

random.uniform(a, b)设定浮点数的范围，一个是上限，一个是下限。

random.randint(-100, 100)随机生一个整数int类型，可以指定这个整数的范围

random.randrange(0, 101, 5)区间选择值

random.choice(seq) 从序列seq中返回随机的元素；

random.getrandbits(n) 以长整型形式返回n个随机位；

random.shuffle(seq[, random]) 原地指定seq序列；

random.sample(seq, n) 从序列seq中选择n个随机且独立的元素；

非均分分布

random.normalvariate()

random.gauss()

random.lognormvariate()

random.triangular()

random.expovariate()

random.paretovariate()

random.vonmisesvariate()

random.betavariate()

random.gammavariate()

random.weibullvariate()

### $from collections import defaultdict, OrderDict, ChainMap, Counter, deque

result = defaultdict(list)

有序字典OrderDict

C = ChainMap(dict\_a, dict\_b) #合并字典 重复key保留

### 迭代器函数 $from itertools import chain, izip, imap, islice, …

与使用列表的代码相比，基于迭代器的算法可以提供更好的内存使用特性。在真正需要数据之前，并不从迭代器生成数据，由于这个原因，不需要将所有数据都同时存储在内存中，这种“懒”处理模型可以减少内存使用，相应地还可以减少交换以及大数据集的其他副作用，从而改善性能

$from itertools import \*

accumulate([1,2,3,4,5], operator.add) => 1 3 6 10 15

chain(‘ABC’, ‘DEF’) => A B C D E F #chain连接多个列表

combinations(‘ABCD’, 2) => AB AC AD BC BD CD … #求列表指定数目的不重复组合

combinations\_with\_replacement(‘ABC’, 2) => AA AB … #重复组合

permutations(‘ABCD’, 2) => AB AC AD BA BC BD … #求列表指定数目的排列

compress(‘ABCDEF’, [1,0,1,0,1,1]) => ACEF #按照真值表筛选元素

cycle(‘ABCD’) => A B C D A B C D … #循环列表

groupby([(0, 0), (1, 0), (0, 1), (1, 1)], lambda x: x[0]) #按照分组函数值对元素分组

=>

[(0, 0), (0, 1)]

[(1, 0), (1, 1)]

islice(‘ABCDEFG’，start, end, step) #切片列表

islice(‘ABCDEFG’，2, None) => CDEFG

### $import re

1、匹配字符

. 匹配任意除换行符，也就是“\n”以外的任何字符。

\ 转义符，改变原来符号含义，后面会有演示。

[ ] 中括号用来创建一个字符集，第一个出现字符如果是^，表示反向匹配。

2、预定义字符集

\d 匹配数字，如：[0-9]

\D 与上面正好相反，匹配所有非数字字符。

\s 空白字符，如：空格，\t\r\n\f\v等。

\S 非空白字符。

\w 单词字符，如：大写A~Z，小写a~z，数字0~9。

\W 非上面这些字符。

[^abc] 匹配不在这些字符类的所有字符

^Hello 表明匹配必须发生在被查找文本开始处

Hello$ 表明被查找文本必须以这个正则表达式的模式结束

3、可选项与重复子模式

｜ 用管道匹配或关系 re.compile(r’Batman|Tina’)

\* 匹配前一个字符0次或无限次数。

+ 匹配前一个字符1次或无限次数。\*

? 匹配前一个字符0次或1次。

{m} 匹配前一个字符m次。

{m,n} 匹配前一个字符m至n次。

compile(模式字符串) 根据正则表达式字符串，创建模式的对象。

search(待查找字符串) 返回第一个匹配项（Match对象）。

split() 根据模式的匹配项来分割字符串。

findall(待查找字符串) 返回所有匹配项(文本列表,不是Match对象)，若有分组，返回元组的列表

sub(new, 待处理字符串) 用将所有匹配项用new替换掉

escape() 将字符串中所有特殊正则表达式字符转义

import re

phoneRegex = re.compile(r'\d{3}-\d{3}-\d{4}') 创建正则表达式对象

matchObj = phoneRegex.search(待查找字符串) 查找字符串,返回Match对象

matchObj.group() 返回实际匹配文本的字符串

利用括号分组,第一对括号是第一组，第二对括号是第2组

phoneRegex = re.compile(r'(\d{3})-(\d{3}-\d{4})')

matchObj = phoneRegex.search(待查找字符串)

matchObj.group(1) 返回第一个分组

matchObj.group(2) 返回第二个分组

matchObj.group() 返回整个匹配文本即\d{3}-\d{3}-\d{4}

areaCode, mainNumber = matchObj.groups() 返回所有分组的元组

Python的正则表达式默认是贪心的，这表示在有二义的情况下，它们会尽可能匹配最长的字符串

采样

import random

# choosing 4 random sample without replacement from a list

Sample = random.sample(data\_list, 4)

# choose 4 random sample with replacement

Sample = radom.choices(data\_list, k=4)

## 单元测试

[https://docs.python.org/3.8/library/unittest.html#](https://docs.python.org/3.8/library/unittest.html)

import unittest

class WidgeTestCase(unittest.TestCase):

def setUp(self): # 类初始化，仅执行一次

self.widget = Widget()

def test\_size(self):

self.assertEuqal(self.widget.size(), (50, 50), ‘incorrect default size’)

def test\_resize(self):

self.widget.resize(100, 150)

self.assertEqual(self.widget.size(), (100, 150), ‘wrong size after resize’)

def teardown(self): # 类收尾工作，仅执行一次

self.widget.dispose()

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

unittest.main(verbosity=2)

## 文件解析

### import pprint

pprint.pprint()美观打印python数据结构

pprint.pformat()返回python数据结构的字符串表示。一般用来保存变量和内容到\*.py文件中，该文件将成为你自己的模块，如果你需要使用存储在其中的变量，就可以导入它

from pprint import pformat

cats = [{'name': 'qzlin', 'desc': 'sleepy'}, {...}]

fileObj = open('myCats.py', 'w')

fileObj.write('cats = ' + pformat(cats) + '\n')

fileObj.close()

### copy and paste

import pyperclip

pyperclip.copy('Hello world')

pyperclip.paste()

### $import json,csv, np.save/load, pickle.dump/load

$import json

list = [“iplaypython”, [1,2,3], {“name”: “xiaoming”}] 列表

//dumps()将Python数据结构转换成json字符串

encoded\_json = json.dumps(list) 将python数据结构进行json格式化编码

with open(‘performance.json’, ‘w’) as ftxt:

json.dump(performance, ftxt, indent=4) 将python数据结构进行json输出

//loads()读取json字符串

decode\_json = json.loads(encoded\_json) #解码python json格式

data\_json = json.loads(open(‘data/input.json’).read()) #读入json文件

print json.dumps(data\_json, indent=4) #漂亮打印json数据

$import csv

读取写入操作csv数据文件

reader()读取csv文件数据的函数方法；

writer()写入csv文件数据的函数方法；

#!/usr/bin/python

import csv

def write\_csv():

with open('data/some.csv', 'wb') as f:

writer = csv.writer(f, delimiter=' ', quotechar='|', quoting=csv.QUOTE\_MINIMAL)

writer.writerow(['a', '1', '2'])

writer.writerow(['b', '2', '3'])

def read\_csv():

with open('data/some.csv', 'rb') as f:

reader = csv.reader(f, delimiter=' ', quotechar='|')

for row in reader:

print ','.join(row)

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

write\_csv()

read\_csv()

$import numpy as np

with open(‘test.npy’, ‘wb’) as f:

np.save(f, data) # data is np.array

with open(‘test.npy’, ‘rb’) as f:

data = np.load(f)

$import pickle

with open(‘test.pickle’, ‘wb’) as f:

pickle.dump(data, f)

with open(‘test.pickle’, ‘rb’) as f:

tics = pickle.load(f)

### Beautiful Soup --HTML and XML 解析

### PDF,Excel,World解析

* **PDF**

方法一：pdf2htmlEX -> BeautifulSoup

方法二：PyPDF2

PyPDF2 is a pure-python PDF library capable of splitting, merging together, cropping, and transforming the pages of PDF files.PyPDF2 is a pure-python PDF library capable of splitting, merging together, cropping, and transforming the pages of PDF files

<https://www.binpress.com/tutorial/manipulating-pdfs-with-python/167>

PyPDF2没有办法从PDF文档中提取图像、图表或其他媒体，但它可以提取文本，并将文本返回为Python字符串

import PyPDF2

pdfFileObj = open('example.pdf', 'rb')

pdfReader = PyPDF2.PdfFileReader(pdfFileObj)

//解密PDF

pdfReader.isEncrypted

pdfReader.decrypt('password')

pdfReader.numPages

pageObj = pdfReader.getPage(0)

//提取文本，可以会出错

pdfObj.extractText()

//拷贝页面，旋转页面，叠加页面，加密页面，

加水印

import os

from pyPdf import PdfFileWriter, PdfFileReader

from reportlab.pdfgen import canvas

def create\_watermark(content, pdf\_watermark):

c = canvas.Canvas(pdf\_watermark)

c.setFont('Courier', 10)

c.saveState()

c.translate(300, 15)

c.drawCentredString(0, 0, content)

c.restoreState()

c.save()

pdf\_watermark = PdfFileReader(file(pdf\_watermark, 'rb'))

return pdf\_watermark

def add\_watermark(pdf\_file, pdf\_watermark):

pdf\_output = PdfFileWriter()

pdf\_input = PdfFileReader(file(pdf\_file, 'rb'))

watermark = pdf\_watermark.getPage(0)

numPages = pdf\_input.getNumPages()

for i in xrange(numPages):

page = pdf\_input.getPage(i)

page.mergePage(watermark)

pdf\_output.addPage(page)

with open('output.pdf', 'wb') as f:

pdf\_output.write(f)

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

pdf\_watermark = create\_watermark('www.site-digger.com', 'watermark.pdf')

add\_watermark('test.pdf', pdf\_watermark)

* **Excel解析**

一个Excel电子表格文档称为一个工作簿。每个工作簿含多个工作表，用户当前查看的表，称为活动表

每个表都有一些列（地址是从A开始的字母）和一些行（地址是从1开始的数字）

import openpyxl

wb = openpyxl.load\_workbook('example.xlsx')

wb.get\_sheet\_names()

sheet = wb.get\_sheet\_name('Sheet3')

sheet.title

activeSheet = wb.get\_active\_sheet()

sheet['A1'].value

sheet.cell(row=1, column=2).value

for rowOfCellObjects in sheet['A1':'C3']:

for cellObj in rowOfCellObjects:

fun(cellObj.coordinate, cellObj.value)

wb.save('exmaple\_copy.xlsx')

from openpyxl.styles import Font, Style

styleOj = Style(font=Font(size=24, italic=True))

sheet['A'].style = styleObj

sheet['A1'] = 'Hello world'

wb.save('styled.xlsx')

设置行高列宽，合并拆分单元格，图表等。。。

* **Word解析**

在最高一层，Document对象表示整个文档，Document对象包含一个Paragraph对象的列表，表示文档中的段落，每个Paragraph对象都包含一个Run对象的列表。

Word文档中的文本不仅仅是字符串，它包含与之相关的字体、大小、颜色和其他样式信息

pip install python-docx

import docx

doc = docx.Document('demo.docx')

paragraphList = doc.paragraphs

paragraphList[0].text

runList = paragraphList[0].runs

runList[0].text

fullText = []

for para in doc.paragraphs:

fullText.append(para.text)

'\n'.join(fullText)

设置Paragraph and Run对象的样式

对于Word文档，有3种类型的样式：段落样式可以应用于Paragraph对象，字符样式可以应用于Run对象，链接样式可以应用于这两种对象

具体见<Python编程快速上手>p256

paraObj = doc.add\_paragraph('...')

paraObj.add\_run('...')

doc.add\_picture('zo.png', width=docx.shared.Inches(1), height=docx.shared.Cm(4))

在处理PDF and Word文档时有很多限制，因为这些格式的本章是很好地展示给人看，而不是让软件易于解析。而Json and CSV文件是设计给计算机使用

## 多线程，多进程，定时任务和GUI自动化

* **多线程 import threading, queue**

计算机的时钟可以调度程序，在特定的时间和日期运行，或定期运行

利用subprocess and threading模块，可以编程按时启动其他程序

不必让所有的代码等待，直到time.sleep()函数完成，你可以使用Python的threading模块，在单独的线程中执行延迟或安排的代码。这个单独的线程将因为time.sleep()调用而暂停。同时，程序可以在原来的线程中做其他工作

import threading, time

print('Start of program')

def takeANap():

time.sleep(5)

print('Wake up!')

threadObj = threading.Thread(target=takeANap)

threadObj.start()

print('End of program.')

向线程的目标函数传递参数

threadObj = threading.Thread(target=print, args=['cats', 'dogs', 'frogs'], kwargs={'sep': '&'})

threadObj.start()

常规参数可以作为一个列表，传递给threading.Thread()中的args关键字参数。关键字参数可以作为一个字典，传递给threading.Thread()中的kwargs关键字参数

获取多线程的结果

import threading

from queue import Queue

data = [[1, 2, 3], [3, 4, 5], [2, 2, 2], [3, 3, 3]]

def job(data, res):

result = np.sum(np.array(data))

res.put(result) #用queue.Queue()保存线程结果

# 多线程执行

res = Queue()

for i in range(len(data)):

t = threading.Thread(target=job, args=(data[i], res))

t.start()

t.join()

#获取多线程结果

result = [res.get() for j in range(len(data))]

* **多进程 import subprocess, multiprocessing**

import multiprocessing as mp

# 多进程执行

res = mp.Queue()

for i in range(len(data)):

t = mp.Process(target=job, args=(data[i], res))

t.start()

t.join()

# 使用进程池

from multiprocessing import cpu\_count

print("CPU的核数为：{}".format(cpu\_count()))

pool = multiprocessing.Pool(processes=4)

result = []

for i in xrange(10):

result.append(pool.apply\_async(func, args=(arg1, arg2)))

pool.close()

pool.join()

[print res.get() for res in result]

import subprocess

subprocess.Popen([r'c:\Windows\notepad.exe', r'c:\hello.txt'])

向Popen()传递一个列表，作为唯一的参数。该列表中的第一个字符串是要启动的程序的可执行文件名，所有后续的字符串将是该程序启动时，传递给该程序的命令行参数。实际上，这个列表将作为被启动程序的sys.argv的值

* **定时任务**

cron job被用于安排那些需要被周期性执行的命令。利用它，你可以配置某些命令或者脚本，让它们在某个设定的时间内周期性地运行。cron 是 Linux 或者类 Unix 系统中最为实用的工具之一。cron 服务（守护进程）在系统后台运行，并且会持续地检查 /etc/crontab 文件和 /etc/cron.\*/目录。它同样也会检查 /var/spool/cron/ 目录。

crontab 是用来安装、卸载或者列出定时任务列表的命令。cron 配置文件则用于驱动 Vixie Cron 的 cron(8) 守护进程。每个用户都可以拥有自己的 crontab 文件

编辑定时任务

$ crontab -e

m h dom mon dow /path/to/command arg1 arg2

m字段：分钟 (0-59)

h字段：小时 (0-23)

dom字段：日期 (1-31)

mon字段：月份 (1-12 [12 代表 December])

dow字段：一周当中的某天 (0-7 [7 或 0 代表星期天])

/path/to/command - 计划执行的脚本或命令的名称

简单的 crontab 示例：

### 每隔 5 分钟运行一次 backupscript 脚本 ##

\*/5 \* \* \* \* /root/backupscript.sh

### 每天的凌晨 1 点运行 backupscript 脚本 ##

0 1 \* \* \* /root/backupscript.sh

### 每月的第一个凌晨 3:15 运行 backupscript 脚本 ##

15 3 1 \* \* /root/backupscript.sh

每隔1分钟运行一次/home/qlin/.../thread-ex.py,并且将output输出到out-e

\*/1 \* \* \* \* /home/qzlin/…/threading-ex.py >> /home/qzlin/…/out-threading-ex.txt

* **GUI自动化**

<python编程快速上手>p352

Pyautogui模块包含一些函数，可以模拟鼠标移动、按键和滚动鼠标滚轮

$pip install python-xlib

$apt-get install scrot python-tk python-dev

$pip install pyautogui

import pyautogui

pyautogui.PAUSE = 1 每个PyAutoGUI函数调用后，暂停一秒

自动防故障功能，鼠标移到屏幕左上角，导致pyautogui.FailSafeException异常

pyautogui.FAILSAFE = True

pyautogui.moveTo(x, y, duration=0.25) 移至(x,y),需0.25秒

pyautogui.moveRel(deltaX, deltaY, duration=0.25) 相对于当前位置

try:

while True:

x, y = pyautogui.position()

positionStr = 'X: ' + str(x).rjust(4) + ' Y: ' + str(y).rjust(4)

print(positionStr, end='')

print('\b' \* len(positionStr), end='', flush=True)

except KeyboardInterrupt:

print('\nDone.')

pyautogui.click(x, y)

pyautogui.dragRel(deltaX, deltaY, duration=0.25)

滚动鼠标：单位的意义在每个操作系统和应用上不一样，滚动发生在鼠标的当前位置。传递正整数表示向上滚动，传递负整数表示向下滚动

pyautogui.scroll(200)

屏幕快照

im = pyautogui.screenshot()

im.getpixel((x, y))

pyautogui.pixelMathesColor(50, 200, (130, 135, 144))

打字

pyautogui.click(x, y)

pyautogui.typewrite('Hello world')

pyautogui.hotkey('ctrl', 'c')

## 数据库

### Sqlite3

SQLite是一个进程中关系数据库，SQLite设计为嵌入在应用中，而不是作为一个单独的数据库服务器程序

$sudo apt-get install sqlite3 安装sqlite3

**todo\_schema.sql**

**sqlite-generate-db.py**

import os

import sqlite3

db\_filename = 'todo.db'

schema\_filename = 'todo\_schema.sql'

db\_is\_new = not os.path.exists(db\_filename)

with sqlite3.connect(db\_filename) as conn:

if db\_is\_new:

print 'Creating schema'

with open(schema\_filename, 'rt') as f:

schema = f.read()

conn.executescript(schema)

else:

print 'Database exists'

**sqlite-query.py**

import sqlite3

db\_filename = 'todo.db'

with sqlite3.connect(db\_filename) as conn:

conn.row\_factory = sqlite3.Row

cursor = conn.cursor()

query = """

select id, priority, details, status, deadline

from task

where project = :project\_name

order by deadline

"""

cursor.execute(query, {'project\_name': 'pymotw'})

for row in cursor.fetchall():

print '%2d {%d} %-25s [%-8s] (%s)' % (row['id'], row['priority'], row['details'], row['status'], row['deadline'])

事务性

with sqlite3.connect(db\_filename) as conn:

try:

cursor = conn.cursor()

cursor.execute("""...""")

cursor.execute("""...""")

execpt Exception, err:

conn.rollback() 回滚

else:

conn.commit() 保存

## 网络数据采集

### 网络请求和响应 --requests

$pip install requests

Requests模块是“人类“用的http模块，类似的模块有urllib，urllib2，httplib，httplib2

#### 7个http动作 --发送请求, 返回Response 对象

r = requests.get('https://github.com/timeline.json')

r = requests.post("http://httpbin.org/post")

r = requests.put("http://httpbin.org/put")

r = requests.delete("http://httpbin.org/delete")

r = requests.head("http://httpbin.org/get")

r = requests.options("http://httpbin.org/get")

#### 为URL传递参数

payload = {'key1': 'value1', 'key2': 'value2'}

r = requests.get("http://httpbin.org/get", params=payload)

r.url == http://httpbin.org/get?key2=value2&key1=value1

#### 定制请求头

url = 'https://api.github.com/some/endpoint'

payload = {'some': 'data'}

headers = {'content-type': 'application/json'}

r = requests.post(url, data=json.dumps(payload), headers=headers)

表单形式的POST请求

payload = {'key1': 'value1', 'key2': 'value2'}

r = requests.post("http://httpbin.org/post", data=payload)

POST一个多部分编码(Multipart-Encoded)的文件

url = 'http://httpbin.org/post'

files = {'file': open('report.xls', 'rb')}

r = requests.post(url, files=files)

Basic Authentication

requests.get('https://api.github.com/user', auth=('user', 'pass'))

OAuth 1 Authentication

auth = OAuth1('YOUR\_APP\_KEY', 'YOUR\_APP\_SECRET',

'USER\_OAUTH\_TOKEN', 'USER\_OAUTH\_TOKEN\_SECRET')

requests.get(url, auth=auth)

Proxies

proxies = {

'http': 'http://165.225.96.34:10015',

'https': ' http://165.225.96.34:10015',

}

requests.get('http://example.org', proxies=proxies)

#### 响应对象

r.headers 响应头

r.status\_code == requests.codes.ok 响应状态码

r.cookies Cookies

r.text 响应内容

r.content 二进制响应内容

from PIL import Image

from StringIO import StringIO

i = Image.open(StringIO(r.content))

r.json JSON响应内容

r.raw 原始响应内容

#### 保存响应内容

with open(filename, 'wb') as fd:

for chunk in r.iter\_content(chunk\_size):

fd.write(chunk)

### 解析HTML and XML --Beautiful Soup

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

$pip install beautifulsoup4

$pip install lxml HTML解析器 速度快

$pip install html5lib 解析方式与浏览器相同

from bs4 import BeautifulSoup

#### 创建BeautifulSoup对象

soup = BeautifulSoup(open("index.html"))

soup = BeautifulSoup("<html>data</html>")

Beautiful Soup将复杂HTML文档转换成一个复杂的树形结构,每个节点都是Python对象,所有对象可以归纳为4种: Tag , NavigableString , BeautifulSoup , Comment .

指定文档解析器

如果仅是想要解析HTML文档,只要用文档创建 BeautifulSoup 对象就可以了，Beautiful Soup会自动选择一个解析器来解析文档.但是还可以通过参数指定使用那种解析器来解析当前文档.

BeautifulSoup 第一个参数应该是要被解析的文档字符串或是文件句柄,第二个参数用来标识怎样解析文档.如果第二个参数为空,那么Beautiful Soup根据当前系统安装的库自动选择解析器,解析器的优先数序: lxml, html5lib, Python标准库.在下面两种条件下解析器优先顺序会变化:

要解析的文档是什么类型: 目前支持, “html”, “xml”, 和 “html5”

指定使用哪种解析器: 目前支持, “lxml”, “html5lib”, 和 “html.parser”

编码

任何HTML或XML文档都有自己的编码方式,比如ASCII 或 UTF-8,但是使用Beautiful Soup解析后,文档都被转换成了Unicode:

Beautiful Soup用了编码自动检测子库来识别当前文档编码并转换成Unicode编码

编码自动检测功能大部分时候都能猜对编码格式,但有时候也会出错.有时候即使猜测正确,也是在逐个字节的遍历整个文档后才猜对的,这样很慢.如果预先知道文档编码,可以设置编码参数来减少自动检查编码出错的概率并且提高文档解析速度.在创建 BeautifulSoup 对象的时候设置 from\_encoding 参数.

#### 标签Tag对象

Tag 对象与XML或HTML原生文档中的tag相同:

soup = BeautifulSoup('<b class="boldest">Extremely bold</b>')

tag = soup.b

tag中最重要的属性: name和attributes

str(tag) tag -> str

tag.name == 'b'

tag.attrs 字典

tag['class']

tag.string == u'Extremely bold'

unicode\_string = unicode(tag.string)

tag的 .contents tag的子节点 （列表）

tag.contents[0]

tag.descendants tag的所有子孙节点 （列表）

tag.strings 子孙字符串 （列表）

tag.stripped\_strings 去除多余空白的子孙字符串 （列表）

tag.parent 元素的父节点

tag.parents 元素的所有父辈节点 (列表）

tag.next\_sibling 和 .previous\_sibling 属性来查询兄弟节点

tag.next\_siblings 和 .previous\_siblings 属性可以对当前节点的兄弟节点迭代输出

tag.get\_text() 方法,这个方法获取到tag中包含的所有文版内容包括子孙tag中的内容

#### Tag查询

find\_all( name , attrs , recursive , text , \*\*kwargs )

soup.a 第一个<a>标签:

soup.find\_all('a') 所有的<a>标签

soup.find\_all(["a", "b"]) 所有<a>标签和<b>标签

soup.find\_all(id='link2')

soup.find\_all("a", attrs={"class": "sister"})

soup.find\_all(text=re.compile("Dormouse"))

soup.select("p nth-of-type(3)") 基于css选择器方法，返回Tag列表

Tag创建与替换

创建一个tag最好的方法是调用工厂方法 BeautifulSoup.new\_tag() :

tag.append(new\_tag)

PageElement.replace\_with() 方法移除文档树中的某段内容,并用新tag或文本节点替代它

PageElement.wrap() 方法可以对指定的tag元素进行包装

#### Tag美观打印

prettify() 方法将Beautiful Soup的文档树格式化后以Unicode编码输出,每个XML/HTML标签都独占一行

#### 常用场景：

从文档中找到所有<a>标签的链接

for link in soup.find\_all('a'):

print(link.get('href'))

# http://example.com/elsie

# http://example.com/lacie

# http://example.com/tillie

从文档中获取所有文字内容

print(soup.get\_text())

# The Dormouse's story

#

# ...

下载所有XKCD漫画

import requests, os, bs4

url ='http://xkcd.com'

os.makedirs('xkcd', exist\_ok=True)

while not url.endswith('#'):

res = requests.get(url)

res.raise\_for\_status() #如果下载文件出错，抛出异常

soup = bs4.BeautifulSoup(res.text)

comicElem = soup.select('#comic img')

if comicElem == []:

print('Could not find comic image.')

else:

comicUrl = 'http:' + comicElem[0].get('src')

res = requests.get(comicUrl)

res.raise\_for\_status()

即使页面是纯文本的，你也需要写入二进制数据，而不是文本数据，目的是为了保存该文本中的"unicode编码"

imageFile = open(os.path.join('xkcd', os.path.basename(comicUrl)), 'wb')

for chunk in res.iter\_content(100\*1000):

imageFile.write(chunk)

imageFile.close()

#Get the Prev button's url

prevLink = soup.select('a[rel="prev"]')[0]

url = 'http://xkcd.com' + prevLink.get('href')

用requests的响应text创建BeautifulSoup会有中英文乱码问题，建议使用如下

import urllib2, bs4

html = urllib2.urlopen(url).read()

soup = bs4.BeautifulSoup(html, 'lxml')

### 模拟浏览器操作

#### import webbrowser

webbrowser.open('http://inventwithpython.com/') 启动浏览器，打开url

selenium模块让Python直接控制浏览器，实际点击链接，填写登录信息, 与Requests and Beautiful Soup相比，Selenium允许你用高级得多的方式与网页交互。但因为它启动了Web浏览器，假如你只是想从网络上下载一些文件，会有点慢，并且难以在后台运行

具体见<Python编程快速上手—让繁琐工作自动化>p210

#### from selenium import webdriver

browser = webdriver.Firefox()

browser.get('http://inventwithpython.com')

查找

browser.find\_elements\_by\_css\_selector(selector)

如果页面上没有元素匹配该方法要查找的元素，selenium模块就会抛出NoSuchElement异常

点击

linkElem = browser.find\_element\_by\_link\_text('Read It Online')

linkElem.click()

填写并提交表单

browser.get('http://gmail.com')

emailElem = browser.find\_element\_by\_id('Email')

passwordElem = browser.find\_element\_by\_id('Passwd')

emailElem.send\_keys('lin.qizhong@gmail.com')

passwordElem.send\_keys('...')

passwordElem.submit() 在任何元素上调用submit()方法，等同于该元素所在表单的submit按钮

from selenium.webdriver.common.keys import Keys

htmlElem = browser.find\_element\_by\_tag\_name('html')

htmlElem.send\_keys(Keys.END)

htmlElem.send\_keys(Keys.HOME) 按下home键，从而页面滚动至顶

## 表单自动填写

<https://selenium-python.readthedocs.io/installation.html>

$apt-get -y install python-pip

$ pip install -U selenium (若不成功)

or

download selenium\*.tar.gz from <https://pypi.python.org/pypi/selenium#downloads>

tar –xvzf selenium\*.tar.gz

cd selenium\*

selenium\*$sudo python setup.py install

若浏览器用ubuntu自带的firefox，selenium 3版本以上需要geckodriver

download geckodriver <https://github.com/mozilla/geckodriver/releases> and mv geckordriver to /usr/bin

# <python基础教程>

python.py -> Python解释器 -> 操作系统 -> 硬件

* **模块和包**

模块（即一个py文件）主要用于定义：在模块作用域里定义类，函数，变量等等，然后**将模块放在可以被解释器找到的路径中**。

引用: 任何Python程序都可以作为模块导入,如math.py

import math; 用import导入模块，然后按照模块.函数的格式使用这个模块的函数

from math import sqrt; 直接使用math模块里的sqrt函数

模块能够明白自己是否是主程序,如Convert.py

def to\_celsius(t):

return (t - 32.0) \* 5.0 / 9.0

**if \_\_name\_\_ == '\_\_main\_\_': #主要是测试代码**

print to\_celsius(80)

如果Convert.py当成main运行,则\_\_name\_\_变成\_\_main\_\_

如果Convert.py被别的模块引用，则\_\_name\_\_变成模块名Convert

如何搜索到模块？

import sys, pprint

pprint.pprint(**sys.path**) #查看Python解释器的搜索目录

探究模块:

import media, pprint

pprint.pprint([n for n in dir(media) if not n.startswith('\_')])

print media.\_\_all\_\_

**help(media)**

print media.\_\_file\_\_ #查看源代码

标准库：

import sys #访问与Python解释器联系紧密的变量和函数 (如：启动外部程序os.startfile(r’…exe’)

import time, random, re

包

相关的模块存储在各自的文件中(.py)，包将模块组织在一起，包就是模块所在的目录。为了让python将其作为包对待，它必须包含一个命令为\_\_init\_\_.py的文件（模块）

如：drawing包

site-packages/drawing/ #包目录

site-packages/drawing/\_\_init\_\_.py #包代码(drawing模块)

site-packages/drawing/colors.py #包中colors模块

site-packages/drawing/shapes.py #包中shapes模块

如何引用？

import drawing #import the drawing package

import drawing.colors #import the colors module,使用时需用drawing.colors.fun()

from drawing import shapes #import the shapes model, 使用时只需用shapes.fun()

* **python基本类型**

类似c语言sys(“pause”), python语言raw\_input(“press<enter>”)

python以:和缩进格式表示block，function…并且没有分号结尾 （C#,java用{}表示）

python也有docstring,模块头，函数头用’’’注释内容‘’’,这样help(模块名)或help（模块名.函数名）就能看到注释文档

字符串:

单引号 ‘hello world’

双引号”hello world”

长字符跨多行:三引号’’’hello

Qzlin’’’

或

‘Hello \

Qzlin’

原始字符串（无转义，类似C#中的@）： r’c:\Program Files\fnord’

unicode字符串: u’林其忠’

toString(): Javascript: “value=” +３　结果是value=3 ; Python: “value”+3 非法，需改为”value” + str(3)

print: 用逗号，要以打印所有类型，如print ‘Age:’, 22

条件测试：false none 0 “” () [] {} 被解释为假. “==”判定同一性(别名)，”is”判定相等性(值相等)

debug： assert （同c#, java）

* **序列**

通用操作

索引: arr[i]

切片: arr[0:4]

连接(记住：同类型才能相加): [1, 2, 3] + [4]

复制多次: ‘python’\*5, [42]\*10

初始值: []空列表, [0]\*10长度为10，值为0的列表，[none]\*10预留长度为10，没有值的列表

成员资格: ‘[low]’ in arr 检查子串是否在长字串

内建函数: len, min, max

**列表**

是异构的,内容可变，除了pop之外，所有方法只会返回一个特殊值None,也就是说只会修改原列表，而不会创建新列表（因为变量是引用）

初始化：arr = [‘Hello’, 1] ;

序列->list: list(‘Hello’)

插入元素: arr[1:1] = [‘world’]

删除元素：arr[i:j] = [] or del arr[i]

List对象的方法: append, count, extend, index, insert, pop, remove, reverse, sort,

List遍历:

for (i, v) in enumerate(arr):

values[i] = 2\*v ;

enumerate(arr) 返回元组(i, value[i]), Python允许多值赋值，即左侧有多个变量，且右侧有相同数量的值，Python就会对它们进行配对，并一次性完成所有赋值操作.

**元组**

创建： 通过逗号分隔值，或通过圆括号如

Tuple0 = ()

Tuple1 = 1，or (1,)

Tuple3 = 1,2,3 or (1, 2, 3)

序列 -> 元组: tuple([1,2,3])

元组操作: 创建元组,访问元组元素

元组的应用: 元组不可变，所以可以在映射（和集合的成员）中当作键使用.元组作为很多内建函数和方法的返回值存在.

**字符串**

格式化：print ‘%s plus %s equals %s’ % (1, 1, 2)

方法：find,join,lower, replace, split, strip, translate,

如:’/’.join((‘’, ‘usr’, ‘bin’, ‘env’)) => ‘/usr/bin/env’

**集合:通过列表或元组定义**

如: set((2,3,4)), set([2, 3, 4]) ，集合的基本操作同数学如: set1 & set2

**字典**

字典的键只能为不可变类型，如整形，浮点型，字符串，或元组

scientists = {‘Newton’:1632, ‘Darwin’:1809}

序列转字典: dict([(‘name’, ‘Gumby’), (‘age’, 42)]

x = []

x[42] = ‘ff’ ; 出错，列表不可以，需要预留空间x = [none] \* 43

x = {}

x[42] = ‘ff’ ; 字典可以，自动添加

方法：clear,copy,fromkeys,get，has\_key,items,iteritems,keys,iterkeys,popitem, setdefault,update,values, itervalues

一般来说，如果试图访问字典中不存在的项时，如d[‘qzlin’]会出错。d.get(‘qzlin’, ‘N/A’)；当使用get访问一个不存在的键时，普通返回是none，可以自定义默认值N/A替换none.

for (key, value) in scientists.iteritems():

…

**常用操作**

序列解包

x,y = y,x 赋值同时进行,这个特性在当函数或者方法返回元组（或者其他序列或可迭代对象）时，这个特性尤其有用。它允许函数返回一个以上的值并且打包成元组，然后通过一个赋值语句很容易进行访问。

key, value = scoundrel.popitem()

并行迭代多个序列

for name, age in zip(names, ages):

…

编号迭代

for index, string in enumerate(strings)：

strings[index] =

列表推导式

是利用其他列表创建新列表（类似于数学术语中的集合推导式）

list1 = [(x,y) for x in range(3) for y in range(3)]

 result = []

for x in range(3):

for y in range(3):

result.append((x,y))

list2 = [x\*x for x in range(10) if x%3==0]

不常用的命令：

pass: 什么都不做，只是占位符

del: 删除变量，或者数据结构的一部分，但是不能用来删除值。

exec: 执行字符串，字符串里是python语句

eval: 执行字符串表达式，并返回结果

性能比较：

import time

def time\_it(search, v, L):

t1 = time.time()

search(v,L)

t2 = time.time()

return (t2-t1)\*1000

* **函数**

同C++一样，函数可以有默认的参数值。

同Java一样，可变参数列表

def max(\*values):

…

max(1,2,3) => 输入参数1,2,3会形成一个元组赋给values

同Java,C#一样，可以用名称对参数进行指定能让我们以任意顺序对参数进行排列。比如一个用于在GUI中显示文本的函数可以接收很多个参数，分别用于指定字体，粗细，顔色，边框等信息。这个时间利用名称对参数进行标记就会显得轻松许多.

* **异常**

同Java,C++

try:

fun()

except exception, e:

…

finally:

…

在做一件事时去处理可能出现的错误，而不是在开始做事前就进行大量的检查，这个策略可以总结为习语“看前就跳(leap before you look)”

import traceback

try:

...

except:

f=open("c:log.txt",'a')

traceback.print\_exc(file=f)

f.flush()

f.close()

* **类**

函数前后双下划线，表示特殊方法，主要表达被python自动调用.

函数前双下划线，表示私有方法

如：

\_\_init\_\_(self,…) 构造器

\_\_del\_\_(self) 析构方法,一般用不上

\_\_inaccessible(self) 私有方法

实现序列和映射 （应用遵寻规则，实现他们基本的行为）

\_\_len\_\_(self), \_\_getitem\_\_(self, key), \_\_setitem\_\_(self, key, value), \_\_delitem\_\_(self, key),\_\_iter\_\_

一般是子类化内建类型list or dict

class SmartList(list):

def \_\_init\_\_(self, \*args):

super(SmartList, self).\_\_init\_\_(\*args)

类似C#属性：隐藏访问器方法，让所有特性看起来一样。

class Rectangle(object):

def \_\_init\_\_(self):

self.width = 0

self.height = 0

def setSize(self, size):

self.width, self.height = size

size = property(getSize, setSize)

r = Rectangle()

r.size = 150,100 #给size赋值时，会通过属性调用setSize方法

拦截对象的所有字段访问，为了在访问字段的时候可以执行代码，如：

class Rectangle:

def \_\_init\_\_(self):

self.width = 0

self.height = 0

def \_\_setattr\_\_(self, name, value):

if name == ‘size’:

self.width, self.height = value

else:

self.\_\_dict\_\_[name] = value

def \_\_getattr\_\_(self, name):

If name == ‘size’:

return self.width, self.height

else:

raise AttributeError

r = Rectangle()

r.size = 100,200 #给对象赋值时，会自动调用\_\_setattr\_\_

(width, height) = r.size #取对象属性值时，会自动调用\_\_getattr\_\_

**class 继承**

\_\_metaclass\_\_ = type #新式类语法 或者用Class Organism(object):

#Organism 继承于object

class Organism:

#所有数据通过构造器

def \_\_init\_\_(self, name, x, y):

self.name = name

self.x = x

self.y = y

self.atoms = []

#类似于Java toString()

def \_\_str\_\_(self):

return ‘(%s, [%s, %s])’ %\

(self.name, self.x, self.y)

#公有方法

def move(self):

…

#私有方法

def \_\_inaccessible(self):

…

#静态方法

@staticmethod

def smeth():

…

python并没有真正的私有化支持，解释器会对上面翻译成

Organism. move(self)

Organism.\_Organism\_\_inaccessible(self)

所以在实际中，

o = Organism()

b.move() #公有方法访问

b.\_Organism\_\_inaccessible() #私有方法访问

# 继承于Organism

class Arthropod(Organism):

def \_\_init\_\_(self, name, x, y, legs):

#老方法

Organism.\_\_init\_\_(self, name, x, y)

#新方法

super(Arthropod, self).\_\_init\_\_(name, x, y)

self.legs = legs

#以下子类与父类同名函数，实现多态性

def \_\_str\_\_(self):

…

def move(self):

..

* **迭代器**

流式处理。 列表（一次性处理）

class Fibs:

def \_\_init\_\_(self):

self.a = 0

self.b = 1

def next(self):

self.a, self.b = self.b, self.a+self.b

return self.a

def \_\_iter\_\_(self):

return self

fibs = Fibs()

for f in fibs:

if f > 1000:

print f

break

* **生成器**

任何包含yield语句的函数称为生成器。每次产生一个值（使用yield语句），函数就会被冻结：即函数停在那点等待被激活。函数被激活后就从停止的那点开始执行.

生成器推导式与列表推导式区别：

列表推导式返回的是整个列表，而生成器推导式返回的是一个一个值，进行流式处理

e.g：

list1 = [(i+2)\*\*2 for i in range(2,27)] #返回的是整个列表

generator = ((i+2)\*\*2 for i in range(2,27)) #返回的是生成器

generator.next() #值会是16

generator.next()

…

递归生成器应用：打印列表中的元素，列表元素可能为列表

#遍历列表中的数字

def flatten(nested):

try:

for sublist in nested:

for element in flatten(sublist):

yield element

except TypeError:

yield nested

list(flatten([ [[1],2], 3, 4, [5, [6,7]], 8]) )

>>>[ 1, 2, 3, 4, 5, 6, 7, 8]

#遍历列表中的所有元素

def flatten(nested):

try:

try:

nested + ''

except TypeError:

pass

else:

raise TypeError

for sublist in nested:

for element in flatten(sublist):

yield element

except TypeError:

yield nested

list1 = ['foo', ['bar', ['baz']]]

list\_flatten = flatten(list1)

print list(list\_flatten)

* **文件**

返回是文件对象

文件迭代器：当文件对象用于for loop时，Python会自动调用readline方法

e.g:

data = open(‘data.txt’, ‘r’)

for line in data:

print len(line.strip())

data.close()

有类似于C# using：

with open("data.txt’") as file:

for line in file:

print len(line.strip())

open函数的第3个参数控制着文件的缓冲。如果参数是0,I/O就是无缓冲的（所有的读写操作都是直接针对硬盘）；如果是1，I/O就是有缓冲的，（意味Python使用内存来代替硬盘，让程序更快，只有使用flush或close时才会更新硬盘上的数据.写入过的文件总是应该关闭，是因为python可能会缓存（出于效率的考虑而把数据临时存储在某处）写入的数据，如果程序因为某些原因崩溃了，那么数据根本就不会被写入文件。

#文件内容如果可以很小，一次性读取所有内容:

f = open(filename)

for line in f.readlines():

process(line)

f.close()

#文件内容很大，流式处理,一行一行读取

import fileinput

for line in fileinput.input(filename):

process(line)

# <Python in Practice>

1. **sequence unpacking and map unpacking**

first, second, \*rest = sequence

等同于first =sequence[0], second=sequence[1], rest=sequence[2:]

Sequence unpacking可以用来传递任意数量的位置参数

map unpacking可以用来传递任意数量的关键字参数

args = (600, 900)

kwargs = dict(copies=2, collate=False)

def print\_args(\*args, \*\*kwargs):

args 是传入位置参数的tuple

kwargs是传入关键字参数的dict

1. **locals() & glocal()**

它囊括了函数执行到该时间点时所定义的一切变量, 返回的字典对所有局部变量的名称与值进行映射.常配合mapping unpacking，用来实例化字符串模板

str.format(\*\*locals()).

字符串模板形参需要的实参从locals()返回的dict传入

重点：

这两个函数主要提供，基于字典的访问局部和全局变量的方式。  
Python使用叫做名字空间的东西来记录变量的轨迹。名字空间只是一个字典，它的键字就是变量名，字典的值就是那些变量的值。实际上，名字空间可以象Python的字典一样进行访问  
每个函数都有着自已的名字空间，叫做局部名字空间，它记录了函数的变量，包括函数的参数  
和局部定义的变量。

每个模块拥有它自已的名字空间，叫做全局名字空间，它记录了模块的变量，包括函数、类、其它导入的模块、模块级的变量和常量。

还有就是内置名字空间，任何模块均可访问它，它存放着内置的函数和异常。  
当一行代码要使用变量 x 的值时，Python会到所有可用的名字空间去查找变量，按照如下顺序：  
a.局部名字空间 - 特指当前函数或类的方法。如果函数定义了一个局部变量 x，Python将使用  
  这个变量，然后停止搜索。  
b.全局名字空间 - 特指当前的模块。如果模块定义了一个名为 x 的变量，函数或类，Python  
  将使用这个变量然后停止搜索。  
c.内置名字空间 - 对每个模块都是全局的。作为最后的尝试，Python将假设 x 是内置函数或变量。

from module import 和 import module之间的不同。使用 import module，模块自身被导入，但是它保持着自已的名字空间，这就是为什么你需要使用模块名来访问它的函数或属性（module.function）的原因。但是使用 from module import，实际上是从另一个模块中将指定的函数和属性导入到你自己的名字空间，这就是为什么你可以直接访问它们却不需要引用它们所来源的模块的原因。

1. **@classmethod and @staticmethod的区别:**

C++中classmethod和staticmethod是一个概念, 类的静态函数

Python中, staticmethod不常用，可以用模块的函数来替代。Python是脚本语言，classmethod的参数class可以是子类 , 也就是说，classmethod可以通过子类来进行重定义。(相当于接口)

二者最大的区别：对于classmethod的参数，需要隐式地传递类名，而staticmethod参数中则不需要传递类名

范例：

class MyClass:

    @classmethod  # classmethod的修饰符

    def class\_method(cls, arg1, arg2, ...):

    @staticmethod  # staticmethod的修饰符

    def static\_method(arg1, arg2, ...):

除了类级别的函数，还有类级别的变量

class MyClass:

    i = 123 # class-level variable

    def \_\_init\_\_(self):

        self.i = 456 # object-level variable

为了清晰地区分上面两个i，最好的办法就是考虑到python中的一切都是object，所以i=123属于class object的，i=456属于class instance object

范例:

def create\_diagram(factory):

diagram = factory.make\_diagram(30, 7)

rectangle = factory.make\_rectangle(4, 1, 22, 5, "yellow")

text = factory.make\_text(7, 3, "Abstract Factory")

diagram.add(rectangle)

diagram.add(text)

return diagram

class DiagramFactory:

@classmethod

def make\_diagram(Class, width, height):

return Class.Diagram(width, height)

@classmethod

def make\_rectangle(Class, x, y, width, height, fill="white",

stroke="black"):

return Class.Rectangle(x, y, width, height, fill, stroke)

@classmethod

def make\_text(Class, x, y, text, fontsize=12):

return Class.Text(x, y, text, fontsize)

class Diagram:

class Rectangle:

class Text:

class SvgDiagramFactory(DiagramFactory):

class Diagram:

class Rectangle:

class Text:

1. **\_\_slots\_\_**

\_\_slots\_\_限定类对象属性范围，同时优化内存(默认情况下，Python用一个dict来存储对象实例的属性, 对一些在”编译”前就知道该有几个固定属性的小class来说，这个dict就有点浪费内存了。而当你把这个小浪费乘上一百万，那可就大不同了。在Python中，你可以在class中设置\_\_slots\_\_，它是一个包含这些固定的属性名的list。这样Python就不会再使用dict，而且只分配这些属性的空间)

动态语言: 当我们定义了一个class，创建了一个class的实例后，我们可以给该实例绑定任何属性和方法, 动态绑定允许我们在程序运行的过程中动态给class加上功能，这在静态语言中很难实现。

class Student(object):

s = Student()

s.name = 'Michael' # 动态给实例绑定一个属性

def set\_age(self, age): # 定义一个函数作为实例方法

self.age = age

from types import MethodType

s.set\_age = MethodType(set\_age, s, Student) # 给实例绑定一个方法

s.set\_age(25) # 调用实例方法

但是，给一个实例绑定的方法，对另一个实例是不起作用的：

为了给所有实例都绑定方法，可以给class绑定方法：

def set\_score(self, score):

self.score = score

Student.set\_score = MethodType(set\_score, None, Student)

给class绑定方法后，所有实例均可调用

但是，如果我们想要限制class的属性怎么办？比如，只允许对Student实例添加name和age属性。为了达到限制的目的，Python允许在定义class的时候，定义一个特殊的\_\_slots\_\_变量，来限制该class能添加的属性：

class Student(object):

\_\_slots\_\_ = ('name', 'age') # 用tuple定义允许绑定的属性名称

s = Student() # 创建新的实例

s.name = 'Michael' # 绑定属性'name'

s.age = 25 # 绑定属性'age'

s.score = 99 # 绑定属性'score',但不在\_\_slots\_\_属性范围内，错误

使用\_\_slots\_\_要注意，\_\_slots\_\_定义的属性仅对当前类起作用，对继承的子类是不起作用的：

除非在子类中也定义\_\_slots\_\_，这样，子类允许定义的属性就是自身的\_\_slots\_\_加上父类的\_\_slots\_\_。

1. **\_\_init\_\_, \_\_new\_\_, and \_\_call\_\_区别**

\_\_new\_\_(cls, \*args, \*\*kwargs)  创建对象时调用，返回当前对象的一个实例;注意：这里的第一个参数是cls即class本身  
\_\_init\_\_(self, \*args, \*\*kwargs) 创建完对象后调用，对当前对象的实例的一些初始化，无返回值,即在调用\_\_new\_\_之后，根据返回的实例初始化；注意，这里的第一个参数是self即对象本身【注意和new的区别】  
\_\_call\_\_(self,  \*args, \*\*kwargs) 如果类实现了这个方法，相当于把这个类型的对象当作函数来使用，相当于 重载了括号运算符

1. **Exec and eval**

eval(str [,globals [,locals ]])函数将字符串str当成有效Python表达式来求值，并返回计算结果。

globals = {'x': 7, 'y': 10, 'birds': ['Parrot', 'Swallow', 'Albatross']}  
locals = { }  
a = eval("3\*x + 4\*y", globals, locals)

exec语句用来执行储存在字符串或文件中的Python语句,结果存在变量中，而变量会更新locals

1. **Type使用**

即查看一个对象的类型(构建类的实例),常见的有int, str, float, 自定义类型.还有一种叫类类型type,如内建类型或用户定义类型的类型，也称为元类metaclass(用来构建类的东西)

type('foo')   => <class 'str'>

class Foo(object):

...     pass

type(Foo) => <class 'type'>

type(name, bases, dict) -> a new type

Foo = type('Foo', (), {})   #等同于创建类Foo

Foo = type('FooBar', (Foo), {'always\_false': always\_false})   #创建子类

glocals()[‘Foo’] = Foo #将创建的类添加到当前模块sys.modules[\_\_name\_\_]中等同于:

setattr(sys.module[\_\_name\_\_], ‘Foo’, Foo)

1. **面向函数编程：提供的函数**

filter(function, sequence)：对sequence中的item依次执行function(item)，将执行结果为True的item组成一个List/String/Tuple（取决于sequence的类型）返回：  
>>> def f(x): return x % 2 != 0 and x % 3 != 0   
>>> filter(f, range(2, 25))   
[5, 7, 11, 13, 17, 19, 23]  
>>> def f(x): return x != 'a'   
>>> filter(f, "abcdef")   
'bcdef'

map(function, sequence) ：对sequence中的item依次执行function(item)，见执行结果组成一个List返回：  
>>> def cube(x): return x\*x\*x   
>>> map(cube, range(1, 11))   
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]  
>>> def cube(x) : return x + x   
...   
>>> map(cube , "abcde")   
['aa', 'bb', 'cc', 'dd', 'ee']  
另外map也支持多个sequence，这就要求function也支持相应数量的参数输入：  
>>> def add(x, y): return x+y   
>>> map(add, range(8), range(8))   
[0, 2, 4, 6, 8, 10, 12, 14]

reduce(function, sequence, starting\_value)：对sequence中的item顺序迭代调用function，如果有starting\_value，还可以作为初始值调用，例如可以用来对List求和：  
>>> def add(x,y): return x + y   
>>> reduce(add, range(1, 11))   
55 （注：1+2+3+4+5+6+7+8+9+10）  
>>> reduce(add, range(1, 11), 20)   
75 （注：1+2+3+4+5+6+7+8+9+10+20）

lambda：这是Python支持一种有趣的语法，它允许你快速定义单行的最小函数，类似与C语言中的宏，这些叫做lambda的函数，是从LISP借用来的，可以用在任何需要函数的地方：   
>>> g = lambda x: x \* 2   
>>> g(3) 

1. **\_\_call\_\_(self)**

对象通过提供\_\_call\_\_(slef, [,\*args [,\*\*kwargs]])方法可以模拟函数的行为，如果一个对象x提供了该方法，就可以像函数一样使用它，也就是说x(arg1, arg2...) 等同于调用x.\_\_call\_\_(self, arg1, arg2) 。  模拟函数的对象可以用于创建防函数(functor) 或代理(proxy)

class DistanceForm(object):

    def \_\_init\_\_(self, origin):

        self.origin = origin

        print "origin :"+str(origin)

    def \_\_call\_\_(self, x):

        print "x :"+str(x)

p = DistanceForm(100)

p(2000)

1. Tuple and namedtuple

Jane = ('Jane',29,'female')  
print 'Field by index:', Jane[0]

Import collections  
Person = collections.namedtuple('Person','name age gender')

Jane = Person(name='Jane',age=29,gender='female')  
print 'Field by Name:', Jane.name

# <Python编程快速上手—让繁琐工作自动化>

* **基础知识**

range(0, 10, 2)将从0数到8,间隔为2

在Python中有一个值称为None,它表示没有值。**None是NoneType数据类型的唯一值（其他编程语言称为null, nil或undefined）**

下标从0开始增长，整数值-1指列表的最后一个下标，-2倒数第二…

del语名将删除列表中下标处的值，表中被删除值后面的所有值，都将向前移动一个下标。

for i in range(len(list)):

...

list[random.randint(0, len(list)-1)] 随机数下标

**列表是可变的**数据类型，它的值可以添加、删除或改变。**字符串和元组是不可变的**。在变量必须保存**可变数据类型**的值时，例如列表或字典，**Python使用引用**。对于**不可变的数据类型**的值，例如字符串、整形或元组，**Python变量就保存值本身**

import copy

copy.copy()复制列表或字典

copy.deepcopy()复制列表 （含嵌套列表）

字典函数

for k in dict.keys():

for v in dict.values():

for k, v in dict.items():

key in dict 检查字典是否含键

dict.get(key, defaultValue) 获取键值，若不存在则返回初始值

dict.setdefault(key, defaultValue) 检查键，若不存在则设置初始值

# <Effective Python>

# <Flask Web开发>

## tutorial

$pip install Flask

Pyramid, Django, and Flask

Flask is a "microframework" primarily aimed at small applications with simpler requirements.

Pyramid and Django are both aimed at larger applications

hello.py

from flask import Flask

from flask import request

app = Flask(\_\_name\_\_ , static\_url\_path='/static')

@app.route("/")

def hello():

return "Hello World!"

@app.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

do\_the\_login()

else:

show\_the\_login\_form()

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

$python hello.py

curl localhost:5000/

常规应用

$sudo pip install virtualenv

flask-app$ virtualenv env

flask-app$ source env/bin/activate

(env)$ pip install flask

(env)$ pip install flask-script

(env)$ gedit hello.py

#!./env/bin/python

from flask import Flask

from flask\_script import Manager

app = Flask(\_\_name\_\_)

manager = Manager(app)

@app.route('/')

def index():

return '<h1>Hello World!</h1>'

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

manager.run()

(env)$ chmod u+x hello.py

默认情况下，Flask开发Web服务器监听localhost上的连接，若让web服务器监听公共网络接口上的连接，允许同网中的其他计算机连接服务器，需要监听所有IP即0.0.0.0

(env)$ ./hello.py runserver --host 0.0.0.0

以shell方式启动hello.py

(env)$ ./hello.py shell

>>> from hello import \*

(env)$ deactivate

在本地上连接

$curl localhost:5000

在连网别的电脑上连接

$curl 192.168.1.12:5000/ 《假定hello.py服务器IP地址是192.168.1.12

## 视图

* **模板**

from flask import Flask, render\_template

@app.route('/user/<name>')

def user(name):

return render\_template('user.html', name=name)

Flask提供的render\_template函数把Jinja2模板引擎集成到了程序中。render\_template函数的第一个参数是模板的文件名。随后的参数都是键值对，表示模板中变量对应的真实值。

变量

特殊占位符，告诉模板引擎这个位置的值从渲染模板时使用的数据中获取

Jinja2能识别所有类型的变量，甚至是一些复杂的类型，例如列表、字典和对象。

<p>A value from a dict: {{ mydict['key'] }} </p>

<p>A value from a list: {{ mylist[myintvar] }} </p>

<p>A value from object: {{ obj.somemethod() }} </p>

过滤器

<div>{{ content | safe }} </div> 显示变量存储的html代码，使用safe过滤器

控制结构

{% if condition %}

<div>content1</div>

{% else %}

<div>content2</div>

{% endif %｝

<ul>

{% for item in items %}

<li>{{ item }}</li>

{% endfor %}

</ul>

复用

宏

macros.html

{% macro render\_comment(comment) %}

<li>{{ comment }}</li>

{% endmacro %}

comments.html

{% include 'macros.html' %}

<ul>

{% for item in items %}

{{ render\_comment(item) }}

{% endfor %}

</ul>

模板继承

base.html

<html>

<head>

{% block head %}

<title>{% block title %}{% endblock %} - My Application</title>

{% endblock %}

</head>

<body>

{% block body %}

{% endblock %}

</body>

</html>

extend.html

{% extends "base.html" %}

{% block title %}

Index

{% endblock %}

{% block head %}

{{ super() }}

<style></style>

{% endblock %}

{% block body %}

<h1>Hello, World!</h1>

{% endblock %}

Extends指令声明这个模板继承自base.html,继承之后，基模板中的3个块被重新定义，模板引擎会将其插入适当的位置。super()调用父模板的内容

* **集成Twitter Bootstrap**

(env)$ pip install flask-bootstrap

from flask\_bootstrap import Bootstrap

bootstrap = Bootstrap(app)

初始化Flask-Bootstrap之后，就可以在程序中使用一个包含所有Bootstrap文件的基模板。这个模板利用Jinja2的模板继承机制，让程序扩展一个具有基本页面的基模板，其中就有用来引入Bootstrap的元素

Flask-Bootstrap中的基模板提供了一个网页框架，引入了Bootstrap中的所有CSS and JavaScript文件

{% extends "bootstrap/base.html" %}

{% block scripts %}

{{ super() }}

<script type="text/javascript" src="my-script.js"></script>

{% endblock %}

{% block title %} my title {% endblock %}

{% block navbar %}... {% endblock %}

{% block content %} ... {% endblock %}

其余块见p25

* **自定义错误页面**

@app.errorhandler(404)

def page\_not\_found(e):

return render\_template('404.html'), 404

@app.errorhandler(500)

def internal\_server\_error(e):

return render\_template('500.html'), 500

404.html

500.html

* **链接**

{% block head %}

{{ super() }}

<link rel="shortcut icon" href="{{ url\_for('static', filename='favicon.ico') }}" type="image/x-icon">

{% endblock %}

url\_for()函数，第一个参数是视图函数名，第二个参数动态部

url\_for('user', name='john', \_external=True)返回的结果

<http://localhost:5000/user/john>

* **静态文件**

默认设置下，Flask在程序根据目录中名为static的子目录中寻找静态文件

* **使用Flask-Moment本地化日期和时间**

要想在服务器上只使用UTC时间，一个优雅的解决方案是：把时间单位发送给Web浏览器，转换成当地时间，然后渲染。Web浏览器可以更好地完成这一任务，因为它能获取用户电脑中的时区和区域设置

(env) $ pip install flask-moment

启动逻辑

from flask.ext.moment import Moment

moment = Moment(app)

from datetime import datetime

@app.route('/')

def index():

return render\_template('index.html', current\_time=datetime.utcnow())

模板

<p>The local date and time is {{ moment(current\_time).format('LLL') }}.</p>

* **Web表单**

(env)$ pip install flask-wtf

app = Flask(\_\_name\_\_)

app.config[ 'SECRET\_KEY' ] = 'hard to guess string' app.config字典存储框架、扩展和程序本身的配置变量

表单类 (具体见p35)

from flask\_wtf import Form

from wtforms import StringField, SubmitField

from wtforms.validators import Required

class NameForm(Form):

name = StringField('What is your name?', validators=[Required()])

submit = SubmitField('Submit')

把表单渲染成HTML

Flask-Bootstrap提供了一个非常高端的辅助函数，可以使用Bootstrap中预先定义了的表单渲染整个Flask-WTF表单

使用Bootstrap默认样式渲染传入的表单

{% import "bootstrap/wtf.html" as wtf %}

{{ wtf.quick\_form(form) }}

在视图函数中处理表单

@app.route('/', methods=['GET', 'POST']

def index():

name = None

form = NameForm()

if form.validate\_on\_submit():

name = form.name.data

form.name.data = ''

return render\_template('index.html', form=form, name=name)

1. 用户第一次访问程序时，服务器会收到一个没有表单数据的GET请求，所以validate\_on\_submit()将返回False. If语句的内容将被跳过，通过渲染模板处理请求，并传入表单对象和值为None的name变量作为参数。用户会看到浏览器中显示了一个表单
2. 用户提交表单后，服务器收到一个包含数据的POST请求。Validate\_on\_submit()会调用name字段上附属的required()验证函数。如果名字不为空，就能通过验证，validate\_on\_submit()返回True.

推荐如下写法：

@main.route('/login', methods=['GET', 'POST'])

def login():

form = NameForm()

if request.method == 'POST':

if form.validate\_on\_submit():

name = form.name.data

form.name.data = ''

return render\_template('user.html', name=name)

else:

return render\_template('login.html', form=NameForm())

* **Flash消息 （相当于toast）**

@app.route('/')

def index():

flash('Oops') 相当于toast内容

return render\_template('index.html')

index.html

{% for message in get\_flashed\_messages() %｝ 相当于Bootstrap中的alert

<div class="alert alert-warning">

<button type="button" class="close" data-dismiss="alert">...</button>

{{ message }}

</div>

{% endfor %}

## 数据库

抽象层：也称为对象关系映射ORM (Object-Relational Mapper)或对象文档映射(Object-Document Mapper, ODM),在用户不知觉的情况下把高层的面向对象操作转换成低层的数据库指令

SQLAlchemy是一个很强大的关系型数据库框架，支持多种数据库后台。SQLAlchemy提供了高层ORM,也提供了使用数据库原生SQL的低层功能

(env) $ pip install flask-sqlalchemy

若出现no module mysqldb,解决办法如下：

在非 Virtualenv 状态下运行 sudo apt-get install libmysqlclient-dev

然后在(env)$pip install mysql-python

MySQL mysql://user:password@hostname/db

Postgres postgresql://user:password@hostname/db

SQLite sqlite:///absolute/path/to/database

from flask\_sqlalchemy import SQLAlchemy

basedir = os.path.abspath(os.pathdirname(\_\_file\_\_))

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///' + os.path.join(basedir, 'data.sqlite')

app.config['SQLALCHEMY\_COMMIT\_ON\_TEARDOWN'] = True

db = SQLAlchemy(app) 表示程序使用的数据库

模型 （类属性对应数据库表中的列）

SQLAlchemy列类型和列选项，见p48

SQLAlchemy关系选项见p49

class Role(db.Model):

\_\_tablename\_\_ = 'roles'

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.String(64), unique=True)

users = db.relationship('User', backref='role')

class User(db.Model):

\_\_tablename\_\_ = 'users'

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(64), unique=True, index=True)

role\_id = db.Column(db.Integer, db.ForeignKey('roles.id'))

数据库操作

(env)$python hello.py shell 以shell方式启动

>>> from hello import db

>>> db.create\_all() 创建表

db.create\_all()

插入行

admin\_role = Role(name='Admin')

user\_role = Role(name='User')

user\_john = User(username='john', role=admin\_role)

通过数据库会话管理对数据库所做的改动

数据库会话也称为事务，数据库会话能保证数据库的一致性

db.session.add\_all([admin\_role, user\_role, user\_john])

db.session.commit()

admin\_role.name = 'Administrator' 修改行

db.session.add(admin\_role)

db.session.commit()

db.session.delete(user\_role) 删除行

db.session.commit()

查询

Flask-SQLAlchemy为每个模型类都提供了query对象

Role.query.all()

User.query.filter\_by(role=user\_role).all()

Role.query.filter\_by(name='User').all()

## 电子邮件

(env) $pip install flask-mail

Flask-Mail连接到SMTP服务器，并把邮件交给这个服务器发送

(env) $ export MAIL\_USERNAME=<Gmail username>

(env) $ export MAIL\_PASSWORD=<Gmail password>

from flask\_mail import Mail, Message

mail = Mail(app)

from threading import Thread

def send\_async\_email(app, msg):

with app.app\_context():

mail.send(msg)

def send\_mail(to, subject, template, \*\*kwargs):

msg = Message(subject, sender, recipients=[to])

msg.body = render\_template(template+'.txt', \*\*kwargs)

msg.html = render\_template(template+'.html', \*\*kwargs)

thr = Thread(target=send\_async\_email, args=[app, msg])

thr.start()

return thr

## 大型程序结构

├── app

│   ├── email.py

│   ├── \_\_init\_\_.py

│   ├── main

│   │   ├── errors.py

│   │   ├── forms.py

│   │   ├── \_\_init\_\_.py

│   │   └── views.py

│   ├── models.py

│   ├── static

│   └── templates

│   ├── 404.html

│   ├── base.html

│   ├── login.html

│   └── user.html

├── config.py

├── manage.py

└── requirements.txt

app/main/\_\_init\_\_.py 导入main目录下的views.py（路由）和errors.py(错误处理程序）

from flask import Blueprint

**main = Blueprint('main', \_\_name\_\_)**

from .import views, errors

app/\_\_init\_\_.py

bootstrap = Bootstrap()

mail = Mail()

moment = Moment()

db = SQLAlchemy()

def create\_app(config\_name):

app = Flask(\_\_name\_\_)

bootstrap.init\_app(app)

mail.init\_app(app)

moment.init\_app(app)

db.init\_app(app)

from .main import main as main\_blueprint

app.register\_blueprint(main\_blueprint) 注册Blueprint, 从而由Blueprint管理路由和错误处理

return app

路由views.py

from .import main

**@main.route('/login')**

def login():

Flask会为蓝本中的全部端点加上一个命名空间，所以视图函数login()注册的端点名是main.login, url\_for('main.login')

视图base.html

<li><a href="{{ url\_for('main.login') }}">Home</a></li>

启动脚本manage.py

app = create\_app(os.getenv('FLASK\_CONFIG') or 'default')

manager = Manager(app)

migrate = Migrate(app, db)

manager.add\_command('db', MigrateCommand)

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

manager.run()

(env)$ pip freeze >requirements.txt 记录所有依赖包及版本号

在新的虚拟环境下

(env) $pip install -r requirements.txt

用户认证

图像处理

json序列化

@app.route('/image')

def get\_image():

gray = cv.imread('../../data/timg.jpeg', cv.IMREAD\_GRAYSCALE)

return jsonify(gray.tolist())

@app.route('/img\_html')

def iimg\_html():

return render\_template('image.html')

image.html

var img = $.ajax({

url: "{{ url\_for('get\_image') }}",

success: function( data ) {

var imgData = ctx.createImageData(data[0].length, data.length);

for (let nRow = 0 ; nRow < imgData.height; ++ nRow) {

for (let nCol = 0 ; nCol < imgData.width; ++ nCol) {

let idx = nRow\*imgData.width + nCol;

imgData.data[idx\*4 + 0] = data[nRow][nCol]; // red

imgData.data[idx\*4 + 1] = data[nRow][nCol]; // green

imgData.data[idx\*4 + 2] = data[nRow][nCol]; // blue

imgData.data[idx\*4 + 3] = 255; // alpha

}

}

ctx.putImageData(imgData,0,0);

}

});

Video Streaming with Flask

<https://blog.miguelgrinberg.com/post/video-streaming-with-flask>

<http://flask.pocoo.org/docs/0.12/patterns/streaming/>

stream < - > generator function

Multipart responses consist of a header that includes one of the multipart content types, followed by the parts, separated by a boundary marker and each having its own part specific content type.

教程

Browser -> web server -> WSGI(e.g. CGI，mod\_python, mod\_wsgi) -> web应用程序

WSGI: Web Server Gateway Interface (PythonWeb服务器网关接口)

网关的作用就是在协议之间进行转换。

类似于Java中的"servlet" API,允许不同web服务器和应用框架基于通用API交互的通用规范

Python Paste - WSGI底层工具集. 包括多线程, SSL和 基于Cookies, sessions等的验证(authentication)库

基本实现

接受HTTP请求、解析HTTP请求、发送HTTP响应

hello.py

def application(environ, start\_response):

start\_response('200 OK', [('Content-Type', 'text/html')])

return '<h1>Hello, web!</h1>'

environ：一个包含所有HTTP请求信息的dict对象；

start\_response：一个发送HTTP响应的函数。

Python内置了一个WSGI服务器，这个模块叫wsgiref

server.py 负责启动WSGI服务器

from wsgiref.simple\_server import make\_server

from hello import application

# 创建一个服务器，IP地址为空，端口是8000，处理函数是application:

httpd = make\_server('', 8000, application)

# 开始监听HTTP请求:

httpd.serve\_forever()

中间件

扩展WSGI应用功能性的方法

def log\_environ(handler):

def \_inner(environ, start\_fn):

pprint.pprint(environ)

return handler(environ, start\_fn)

return \_inner

def wrap\_query\_params(handler):

def \_inner(environ, start\_fn):

qs = environ.get('QUERY\_STRING')

environ['QUERY\_PARAMS'] = urlparse.parse\_qs(qs)

return handler(environ, start\_fn)

return \_inner

# Applied from bottom to top on the way in, then top to bottom on the way out

MIDDLEWARES = [wrap\_query\_params,log\_environ,handle\_error]

# wrap\_query\_params(log\_environ(handler))

app = reduce(lambda h, m: m(h), MIDDLEWARES, handler)

Flask

<http://docs.jinkan.org/docs/flask/tutorial/index.html>

<http://docs.jinkan.org/docs/flask/index.html>

Flask 依赖两个外部库： Jinja2 模板引擎和 Werkzeug WSGI 工具集

路由

通过Python的装饰器在内部自动地把URL和函数给关联起来Flask通过Python的装饰器在内部自动地把URL和函数给关联起来

Flask 的 URL 规则基于 Werkzeug 的路由模块。这个模块背后的思想是基于 Apache

\*.html

url: "{{ url\_for('get\_image', filename=filename) }}"

controller.py

@app.route('/api/v1.0/image')

def get\_image():

filename = str(request.args.get("filename", ""))

Request 请求对象

<http://docs.jinkan.org/docs/flask/api.html#flask.request>

req.form 一个包含解析过的从 POST 或 PUT 请求发送的表单对象的 MultiDict

req.args 一个包含解析过的查询字符串（ URL 中问号后的部分）内容的 MultiDict

req.cookies 一个包含请求中传送的所有 cookie 内容的 dict

req.headers 进入请求的标头存为一个类似字典的对象

req.data 如果进入的请求数据是 Flask 不能处理的 mimetype ，数据将作为字符串存于此

req.files 一个包含 POST 和 PUT 请求中上传的文件的 MultiDict 。每个文件存储为 FileStorage 对象。其基本的行为类似你在 Python 中见到的标准文件对象，差异在于这个对象有一个 save() 方法可以把文件存储到文件系统上

req.environ 底层的 WSGI 环境

req.method 当前请求的 HTTP 方法

req.path, .base\_url, .script\_root, .url, .url\_root

req.is\_xhr 当请求由 JavaScript 的 XMLHttpRequest 触发时，该值为 True 。 这只对支持 X-Requested-With 标头并把该标头设置为 XMLHttpRequest 的库奏效。这么做的库有 prototype 、 jQuery 以及 Mochikit 等更多

req.json parsed json data or none

req.max\_content\_length

response 响应对象

Flask 把返回值转换为响应对象的逻辑是这样：

1. 如果返回的是一个合法的响应对象，它会从视图直接返回。
2. 如果返回的是一个字符串（如render\_template()），响应对象会用字符串数据和默认参数创建。（字符串为主体的、状态码为 200 OK(or other)、 MIME 类型是text/html (or other) 的响应对象）
3. 如果返回的是一个元组，且元组中的元素可以提供额外的信息。这样的元组必须是 (response, status, headers) 的形式，且至少包含一个元素。 status 值会覆盖状态代码， headers 可以是一个列表或字典，作为额外的消息标头值。
4. 如果上述条件均不满足， Flask 会假设返回值是一个合法的 WSGI 应用程序，并转换为一个请求对象。

res = make\_response(render\_template(...))

res.set\_cookie('username', 'the username')

res.headers

res.status

res.mimetype

配置

<http://docs.jinkan.org/docs/flask/config.html>

应用会需要某种配置。你可能会需要根据应用环境更改不同的设置，比如切换调试模式、设置密钥、或是别的设定环境的东西。

app = Flask(\_\_name\_\_)

app.config['DEBUG'] = True

app.config['JSON\_AS\_ASCII'] = False

app.config['JSONIFY\_PRETTYPRINT\_REGULAR'] = False

配置文件其实是 Python 文件。只有大写名称的值才会被存储到配置对象中

实例文件夹

操作系统有进程当前工作目录的概念。不幸的是，你在 Web 应用中不能依赖此概念，因为你可能会在相同的进程中运行多个应用。为此，提供了 app.root\_path 属性以获取应用的路径。配合 os.path 模块使用，轻松可达任意文件

在 Flask 0.8 中，引入了 Flask.instance\_path 并提出了“实例文件夹” 的新概念

app = Flask(\_\_name\_\_, instance\_path='/path/to/instance/folder')

如果 instance\_path 参数没有赋值，会使用下面默认的位置:

未安装的模块:

/myapp.py

/instance

已安装的包或模块:

$PREFIX/lib/python2.X/site-packages/myapp

$PREFIX/var/myapp-instance

上传文件

<http://docs.jinkan.org/docs/flask/patterns/fileuploads.html#uploading-files>

<https://pythonhosted.org/Flask-Uploads/>

重定向和错误

redirect(url\_for('login'))

@app.errorhandler(404)

def page\_not\_found(error):

return render\_template('page\_not\_found.html'), 404

默认情况下，错误代码会显示一个黑白的错误页面。如果你要定制错误页面， 可以使用 errorhandler() 装饰器:

会话

from flask import Flask, session

## OpenAPI via connextion

<https://connexion.readthedocs.io/en/latest/> #spec-first and api-first development

<https://realpython.com/flask-connexion-rest-api/>

On top of Flask, install Connexion to handle the HTTP requests

(venv) $ pip install "connexion[swagger-ui]==2.14.1"

add REST API endpoint, you’ll use Connexion. The Connexion module allows a Python program to use the **OpenAPI** specification with Swagger. The OpenAPI Specification is an API description format for REST APIs and provides a lot of functionality, including: 1) Validation of input and output data to and from your API. 2) Configuration of the API URL endpoints and the expected parameters

**swagger.yml # OpenAPI specification**

openapi: 3.0.0

info:

title: "RP Flask REST API"

description: "An API about people and notes"

version: "1.0.0"

servers:

- url: "/api" # http://localhost:8000/api

paths:

/people: # http://localhost:8000/api/people

get:

operationId: "people.read\_all" # function for corresponding api

tags: # group the operations in the UI

- "People"

summary: "Read the list of people" # UI display text

responses:

"200":

description: "Successfully read people list"

**app.py # Add Connexion to the App**

from flask import render\_template

import connexion

app = connexion.App(\_\_name\_\_, specification\_dir="./") # internally, create Flask app

app.add\_api("swagger.yml")

@app.route("/")

def home():

return render\_template("home.html")

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host="0.0.0.0", port=8000, debug=True)

**people.py**

from datetime import datetime

PEOPLE = {

"Fairy": {

"fname": "Tooth",

"lname": "Fairy",

"timestamp": datetime.now().strftime(("%Y-%m-%d %H:%M:%S"))

}

}

def read\_all():

return list(PEOPLE.values())

(venv) $python app.py # start server

Open browser: <http://localhost:8000/api/people>

Flask app knows what to serve based on OpenAPI specification in swagger.yml. Additionally, Connexion uses swagger.yml to create API documentation for you.

localhost:8000/api/ui # API documentation

select api -> try it out -> execute

Using OpenAPI with the Swagger UI offers a nice, clean way to create the API URL endpoints.

**swagger.yml # API for create/get/update/delete**

components:

**schemas**:

Person:

type: object # data type of the schema

required:

- lname

properties:

fname:

type: string

lname:

type: string

**parameters**:

lname:

name: lname

description: Last name of the person to get

in: path

required: True

schema:

type: string

paths:

/people:

post:

operationId: people.create

tags:

- People

summary: Create a person

requestBody:

description: Person to create

required: True

content:

application/json:

**schema:**

**x-body-name: person** # 默认为body

**$ref: ‘#/components/schemas/Person’**

responses:

‘201’:

description: Successfully created person

/people/{lname}:

get:

operationId: "people.read\_one"

tags:

- People

summary: "Read one person"

**parameters:**

**- $ref: "#/components/parameters/lname"**

responses:

"200":

description: "Successfully read person"

put:

tags:

- People

operationId: "people.update"

summary: "Update a person"

**parameters:**

**- $ref: "#/components/parameters/lname"** # as keyword arguments in handler functions

responses:

"200":

description: "Successfully updated person"

requestBody:

content:

application/json:

**schema:**

**x-body-name: "person"**

**$ref: "#/components/schemas/Person"**

delete:

tags:

- People

operationId: "people.delete"

summary: "Delete a person"

**parameters:**

**- $ref: "#/components/parameters/lname"**

responses:

"204":

description: "Successfully deleted person"

people.py

from flask import abort, make\_response

def create(person):

lname = person.get("lname")

fname = person.get("fname", "")

if lname and lname not in PEOPLE:

PEOPLE[lname] = {"lname": lname, "fname": fname, "timestamp": get\_timestamp()}

return PEOPLE[lname], 201

else:

abort(406, f"Person with last name {lname} already exists") # send an error message

def read\_one(lname):

if lname in PEOPLE:

return PEOPLE[lname]

else:

abort(404, f"Person with last name {lname} not found")

def update(lname, person):

if lname in PEOPLE:

PEOPLE[lname]["fname"] = person.get("fname", PEOPLE[lname]["fname"])

PEOPLE[lname]["timestamp"] = get\_timestamp()

return PEOPLE[lname]

else:

abort(404, f"Person with last name {lname} not found")

def delete(lname):

if lname in PEOPLE:

del PEOPLE[lname]

return make\_response(f"{lname} successfully deleted", 200)

else:

abort(404, f"Person with last name {lname} not found")

client access to OpenAPI: <https://realpython.com/api-integration-in-python/>

HTTP method API endpoint Description

GET <api>/patient Get a list of patient.

GET <api>/patient/<patient\_id> Get a single patient.

POST <api>/patient Create a new patient.

PUT <api>/patient/<patient\_id> Update a patient.

PATCH <api>/patient/<patient\_id> Partially update a patient.

DELETE <api>/patient/<patient\_id> Delete a patient.

Code range Category

2xx Successful operation

3xx Redirection

4xx Client error

5xx Server error

Code Meaning Description

200 OK The requested action was successful.

201 Created A new resource was created.

202 Accepted The request was received, but no modification has been made yet.

204 No Content The request was successful, but the response has no content.

400 Bad Request The request was malformed.

401 Unauthorized The client is not authorized to perform the requested action.

404 Not Found The requested resource was not found.

415 Unsupported Media Type The request data format is not supported by the server.

422 Unprocessable Entity The request data was properly formatted but contained invalid or missing data.

500 Internal Server Error The server threw an error when processing the request.

1. # swagger

localhost:8000/api/ui # API documentation

select api -> try it out -> execute

1. $ python -m pip install requests # send HTTP requests via requests

import requests

api\_url = <http://localhost:8000/api/people>

# get

response = requests.get(api\_url)

response.json() # python dict

response.status\_code # 200

response.headers["Content-Type"] # 'application/json; charset=utf-8'

# post

patient = {"fname": “Qizhong”, "lname": "Lin"}

response = requests.post(api\_url, json=patient)

headers = {"Content-Type":"application/json"}

response = requests.post(api\_url, data=json.dumps(patient), headers=headers)

response.status\_code # 201

# put

response = requests.put(api\_url, json=patient)

response.status\_code # 200

# patch

patient = {"fname": “Qizhong”}

response = requests.patch(api\_url, json=patient)

response.status\_code # 200

# delete

response = requests.delete(api\_url)

response.status\_code # 204

1. curl

$ curl -i <http://localhost:8000/api/people> # get

$ curl -i [http://localhost:8000/api/people \](http://localhost:8000/api/people%20\) # post

-X POST \

-H 'Content-Type: application/json' \

-d '{"fname": “Qizhong”, "lname": "Lin"}'

**swagger-editor:** to design, define and document RESTful APIs in the Swagger Specification

<https://swagger.io/docs/open-source-tools/swagger-editor/>

<https://editor.swagger.io/>

<https://github.com/swagger-api/swagger-editor>

**swagger-codegen:** to build server stubs and client SDKs directly from a Swagger defined RESTful API

<https://swagger.io/docs/open-source-tools/swagger-codegen/>

<https://github.com/swagger-api/swagger-codegen>

**swagger-cli:** 主openapi文件引用多文件

<https://github.com/APIDevTools/swagger-cli>

一个主openapi文件引用多文件，生成一个完整的文件

The Swagger and OpenAPI specs allows you to split your API definition across multiple files using $ref pointers to reference each file. You can use the swagger-cli bundle command to combine all of those referenced files into a single file, which is useful for distribution or interoperation with other tools.

install node.js

$ npm install -g @apidevtools/swagger-cli # install swagger-ui

$ swagger-cli validate [options] <file> # Validate an API

Options:

--no-schema Do NOT validate against the Swagger/OpenAPI JSON schema

--no-spec Do NOT validate against the Swagger/OpenAPI specification

$ swagger-cli bundle [options] <file> # Combine Multiple Files

Options:

-o, --outfile <file> The output file

-r, --dereference Fully dereference all $ref pointers

-f, --format <spaces> Formats the output using the given number of spaces, default is 2 spaces

-t, --type <filetype> Defines the output file type. The valid values are: json, yaml, default is JSON

-w, --wrap <column> Set the line length for YAML strings, default is no wrapping

# PyQt

QtCore： Core non-GUI classes used by other modules

QtGui： Graphical user interface components

QtMultimedia： Classes for low-level multimedia programming

QtNetwork： Classes for network programming

QtOpenGL： OpenGL support classes

QtScript： Classes for evaluating Qt Scripts

QtSql： Classes for database integration using SQL

QtSvg： Classes for displaying the contents of SVG files

QtWebKit： Classes for rendering and editing HTML

QtXml： Classes for handling XML

QtAssistant： Support for online help

QtDesigner： Classes for extending Qt Designer

QLabel： Used to display text or image

QLineEdit： Allows the user to enter one line of text

(QDateEdit, QTimeEdit, QDateTimeEdit, QLineEdit, …)

QTextEdit： Allows the user to enter multi-line text

QPushButton： A command button to invoke action

QRadioButton： Enables to choose one from multiple options

QCheckBox： Enables choice of more than one options

QSpinBox： Enables to increase/decrease an integer value

(QDoubleSpinbox, …)

QComboBox： Provides a dropdown list of items to select from

(QListBox, …)

QScrollBar： Enables to access contents of a widget beyond display aperture

QSlider： Enables to change the bound value linearly.

QToolBar： Usually at top of QMainWindow or floating. Contains action buttons

(QStatusBar, QMenuBar= QMenu+ QMenu…)

QListView： Provides a selectable list of items in ListMode or IconMode

(QTableView, QTreeView, QGraphicsView)

QPixmap： Off-screen image representation for display on QLabel or QPushButton object

QDialog： Modal or modeless window which can return information to parent window

QMessageBox: These dialogs can be used for a variety of purposes and customized in many ways to display different messages, icons and buttons.

QProgressBar: Progress bars is a great way to visualize progression of computer operations such as file transferring, downloading, uploading, copying etc.

QHBoxLayout

QVBoxLayout

QGridLayout

QTabWidge

APP Example :

class MainWindow(QMainWindow):

def \_\_init\_\_(self, \*args, \*\*kwargs):

super(MainWindow, self).\_\_init\_\_(\*args, \*\*kwargs)

self.setWindowTitle("My Awesome App")

layout = QHBoxLayout()

…

widget = QWidget()

widget.setLayout(layout)

self.setCentralWidget(widget)

## UI Component

Qt Designer -> \*.ui

$pyuic4 -x \*.ui -o \*.py

<https://build-system.fman.io/pyqt5-tutorial>

<https://build-system.fman.io/linux-gui-programming>

**QLabel**

QLabel.setBuddy(...)

class CustomLabel (QLabel):

clicked=pyqtSignal()

def \_\_init\_\_(self, parent=None):

CustomLabel.\_\_init\_\_(self, parent)

def mousePressEvent(self, ev):

self.clicked.emit()

widget = **QCheckBox**()

widget.setCheckState(Qt.Checked)

widget.stateChanged.connect(self.show\_state)

def show\_state(self, s):

print(s == Qt.Checked)

widget = **QComboBox**()

widget.addItems(["One", "Two", "Three"])

# The default signal from currentIndexChanged sends the index

widget.currentIndexChanged.connect( self.index\_changed )

def index\_changed(self, i): # i is an int

print(i)

widget = **QLineEdit**()

widget.setMaxLength(10)

widget.setPlaceholderText("Enter your text")

widget.returnPressed.connect(self.return\_pressed)

widget.selectionChanged.connect(self.selection\_changed)

widget.textChanged.connect(self.text\_changed)

widget.textEdited.connect(self.text\_edited)

spinBox = **QSpinBox**()

spinBox.setRange(1, 400)

spinBox.setPrefix(“:”)

spinBox.setSuffix(" %")

spinBox.setValue(10)

spinBox.setToolTip("zoom the image")

spinBox.setStatusTip(spinBox.toolTip())

spinBox.valueChanged.connect(showImage)

def showImage(value):

pass

Dialogs and Alerts

They are commonly used for file Open/Save, settings, preferences, or for functions that do not fit into the main UI of the application. They are small modal (or blocking) windows that sit in front of the main application until they are dismissed

dlg = QDialog(self)

dlg.setWindowTitle("HELLO!")

dlg.exec\_()

**QAction** — Toolbars, Menus & keyboard shorts

<https://p.yusukekamiyamane.com/> download icons

1. define QAction, Each QAction has names, status messages, icons and signals that you can connect to (and much more).
2. add this action to both the menu and the toolbar

toolbar = **QToolBar**("My main toolbar")

toolbar.setIconSize(QSize(16,16))

self.addToolBar(toolbar)

self.setStatusBar(**QStatusBar**(self))

# You must also pass in any QObject to act as the parent for the action — here we're passing self as a reference to our main window

def createAction(self, text, slot=None, shortcut=None, icon=None, tip=None, checkable=False, signal=”triggered()”):

action = QAction(“Your button", self)

if icon is not None:

action.setIcon(**QIcon**(":/%s.png" % icon))

if shortcut is not None;

action.setShortCut(shortcut)

if tip is not None:

action.setStatusTip(tip)

action.setStatusTip(tip)

if slot is not None:

self.connect(action, SIGNAL(signal), slot)

if checkable:

action.setCheckable(True)

return action

button\_action = self.createAction(“&New…”, self.fileNew, QKeySequence.New, “filenew”, “Create an image file”)

toolbar.addAction(button\_action)

toolbar.addSeparator()

toolbar.addWidget(QLabel("Hello"))

def fileNew (self, s):

print("click", s)

QSplitter

class MainWindow(QMainWindow):

def \_\_init\_\_(self, parent=None):

super(MainWindow, self).\_\_init\_\_(parent)

mainSplitter = **QSplitter**(Qt.Horizontal)

imageSplitter = Qplitter(Qt.Vertical)

imageSplitter.addWidget(imagelistViewer)

imageSplitter.addWidget(form)

imageSplitter.setStretchFactor(0, 1)

imageSplitter.setStretchFactor(1, 2)

mainSplitter.addWidget(imageViewer)

mainSplitter.addWidget(imageSplitter)

mainSplitter.setStretchFactor(0, 3)

mainSplitter.setStretchFactor(1, 1)

setCentralWidget(mainSplitter)

QBoxLayout

QBoxLayout class lines up the widgets vertically or horizontally. Its derived classes are **QVBoxLayout** (for arranging widgets vertically) and **QHBoxLayout** (for arranging widgets horizontally).

layout.setContentsMargins(0,0,0,0)

layout.setSpacing(20)

**QGridLayout**

A GridLayout class object presents with a grid of cells arranged in rows and columns. The class contains addWidget() method. Any widget can be added by specifying the number of rows and columns of the cell.

QFormLayout

**QFormLayout** is a convenient way to create two column form, where each row consists of an input field associated with a label. As a convention, the left column contains the label and the right column contains an input field.

QLayout (=QWidget + Qlayout) -> **QWidget**

window = QWidget()

window.setAutoFillBackground(True) # auto fill its background with the window cooler

palette = window.palette()

palette.setColor(QPalette.Window, QColor(color))

window.setPalette(palette)

layout = QVBoxLayout()

layout.addWidget(…)

layout.addLayout(…)

window.setLayout(layout)

Graphics View Framework

<https://doc.qt.io/qt-5/graphicsview.html>

**QGraphicsItem**

QGraphicsRectItem, QGraphicsEllipseItem, QGraphicsTextItem

QGraphicsItem supports the following features:

* Mouse press, move, release and double click events, as well as mouse hover events, wheel events, and context menu events.
* Keyboard input focus, and key events
* Drag and drop
* Grouping, both through parent-child relationships, and with QGraphicsItemGroup
* Collision detection

**QGraphicsScene** = QGraphicsItem + QGraphicsItem + ...

event propagation architecture:

If the scene receives a mouse press event at a certain position, the scene passes the event on to whichever item is at that position.

item states, such as item selection and focus.

You can select items on the scene by calling QGraphicsScene::setSelectionArea(), passing an arbitrary shape. => QGraphicsScene::selectedItems()

**QGraphicsView**

provides the view widget, which visualizes the contents of a scene. You can attach several views to the same scene, to provide several viewports into the same data set

The view receives input events from the keyboard and mouse, and translates these to scene events (converting the coordinates used to scene coordinates where appropriate)

the view can transform the scene's coordinate system

QGraphicsView::transform()

three coordinate systems in Graphics View:

* Item coordinates

**Items live in their own local coordinate system**.

if you receive a mouse press or a drag enter event, the event position is given in item coordinates.

Child coordinates are relative to the parent's coordinates, Because items' position and transformation are relative to the parent, child items' coordinates are unaffected by the parent's transformation, although the parent's transformation implicitly transforms the child.

Each item on the scene has a scene position and bounding rectangle (QGraphicsItem::scenePos(), QGraphicsItem::sceneBoundingRect()), in addition to its local item pos and bounding rectangle.

* scene coordinates (corresponds to QPainter’s logical coordinates
* view coordinates

View coordinates are the coordinates of the widget. Each unit in view coordinates corresponds to one pixel

**The top left corner of QGraphicsView's viewport is always (0, 0), and the bottom right corner is always (viewport width, viewport height)**. All mouse events and drag and drop events are originally received as view coordinates, and you need to map these coordinates to the scene in order to interact with items.

map coordinates and arbitrary shapes for **view <-> scene <-> item, item <-> item**

QGraphicsView::mapToScene()

QGraphicsView::mapFromScene()

QGraphicsItem::mapToParent()

QGraphicsItem::mapFromParent()

QGraphicsItem::mapToItem()

QGraphicsItem::mapFromItem()

affine transformations

By applying a transformation to the view, you can easily add support for common navigation features such as zooming and rotating.

## styles

**built-in styles**: 'Fusion', 'Windows', 'WindowsVista' (Windows only) and 'Macintosh' (Mac only).

Qt Style Sheets support all the selectors defined in CSS2.

**Style sheet cascading**: Style sheets can be set on the QApplication, on parent widgets, and on child widgets. An arbitrary widget's effective style sheet is obtained by merging the style sheets set on the widget's ancestors (parent, grandparent, etc.), as well as any style sheet set on the QApplication.

app.setStyle('Fusion')

app.style()

**# change the appearance via style sheets**

app.setStyleSheet("QPushButton { margin: 10ex; }")

widget.style() #return QStyle, which is a wrapper “style sheet” style

Style Rules

<https://doc.qt.io/qt-5/stylesheet-syntax.html>

<https://doc.qt.io/qt-5/stylesheet-reference.html#list-of-properties>

QPushButton { color: red }

The rule specifies that QPushButton and its subclasses (e.g., MyPushButton) should use red as their foreground color.

# change color of button text, same to others

palette = QPalette()

palette.setColor(QPalette.ButtonText, Qt.red)

app.setPalette(palette)

# change font

widget = QLabel("Hello")

font = widget.font()

font.setPointSize(30)

widget.setFont(font)

## PyQt提供了两种通信机制：

底层事件处理机制，类似于所有其他GUI库提供的机制

class Widget(QWidget):

def **paintEvent**(self, event):

painter = QPainter(self)

painter.setRenderHint(QPainter.TextAntialiasing)

painter.drawText(self.rect(), Qt.AlignBottom|Qt.AlighHCenter, self.message)

...

def **resizeEvent**(self, event):

oldSize = QResizeEvent.oldSize()

newSize = event.size()

newSize.width()

newSize.height()

self.message = fun(oldSize, newSize)

self.update() #触发绘制事件

# QKeyEvent.key()返回所按下键的编码

特殊键： Qt.Key\_Home, Qt.Key\_PageUp, Qt.Key\_PageDown, Qt.Key\_End,

Qt.Key\_Up, Qt.Key\_Right, Qt.Key\_Down, Qt.Key\_Left

# QKeyEvent.modifiers()返回一个位标识符，以反映Shift, Ctrl and Alt的状态

def **keyPressEvent**(self, event):

self.key = QString()

if event.key() == Qt.Key\_Home:

self.key = "Home"

elif event.key() == Qt.Key\_End:

self.key = "End"

elif event.key() == Qt.Key\_PageUp:

if event.modifiers() & Qt.ControlModifier:

self.key = "Ctrl+PageUp"

else:

self.key = "PageUp"

elif event.key() == Qt.Key\_PageDown:

if event.modifiers() & Qt.ControlModifier:

self.key = "Ctrl+PageDown"

else:

self.key = "PageDown"

elif Qt.Key\_A <= event.key() <= Qt.Key\_Z:

if event.modifiers() & Qt.ShiftModifier:

self.key = "Shift+"

self.key += event.text()

if self.key:

self.key = QString(self.key)

self.update()

else:

QWidget.keyPressEvent(self, event)

def **mouseDoubleClickEvent**(self, event):

self.justDoubleClicked = True

self.update()

def **mousePressEvent**(self, event):

...

def **mouseReleaseEvent**(self, event):

if self.justDoubleClicked:

self.justDoubleClicked = False

else:

self.setMouseTracking(not self.hasMouseTracking())

if self.hasMouseTracking():

self.text = "mouse tracking is on, try moving the mouse, single click to switch it off"

else:

self.text = "mouse tracking is off, single click to switch it on"

self.update()

def **mouseMoveEvent**(self, event):

if not self.justDoubleClicked:

# event.pos().x(), event.pos().y() in widget coords

# globalPos.x(), globalPos.y() in screen coords

globalPos = self.mapToGlobal(event.pos())

self.update()

def **wheelEvent**(self, event):

...

信号和槽Signals & Slots

events emit ‘signal’ ----connect----'slot' (callable python function)

* 信号与槽可以是多对多的关系. 如一个信号可以连接多个槽， 一个信号可以连接另一个信号， 一个槽可以监听多个信号。
* 信号与槽的连接方式可以是同步连接，也可以是异步连接。
* 信号与槽的连接可能会跨线程。
* 信号可能会断开。

内置Signals & Slots

# default signal by UI component

在事件模型，有三个参与者：事件源，事件对象和事件目标

Example 1:

想知道哪一个组件是信号发送者？PyQt5拥有了sender()方法来解决这个问题

class Widget(QWidget):

def \_\_init\_\_(self):

super(Widget, self).\_\_init\_\_()

self.layout =QGridLayout(self)

for row in xrange(3):

for col in xrange(3):

button = QtGui.QPushButton("Button %d-%d" % (row,col))

button.**clicked.connect**(self.buttonClicked)

self.layout.addWidget(button, row, col)

def buttonClicked(self):

button = **self.sender()**

idx = self.layout.indexOf(button)

location = self.layout.getItemPosition(idx)

print "Button", button, "at row/col", location[:2]

example 2:

self.fromComboBox.currentIndexChanged.connect(self.updateUi)

self.toComboBox.currentIndexChanged.connect(self.updateUi)

self.fromSpinBox.valueChanged.connect(self.updateUi)

def updateUi(self):

to = self.toComboBox.currentText()

value = self.fromSpinBox.value()

...

self.button1.clicked.connect(self.clicked)

self.button2.clicked.connect(self.clicked)

def clicked(self):

button = self.sender()

if button is not None or not isinstance(button, QPushButton):

return

button.text()

自定义Signals & Slots

class QTypeSignal(QObject):

#定义一个信号,可以有参数传递

**sendmsg=pyqtSignal(config)**

def \_\_init\_\_(self):

super(QTypeSignal, self).\_\_init\_\_()

def run( self ):

**self.sendmsg.emit(config)**

def on\_response( config ):

pass

send = QTypeSignal()

send.sendmsg.connect(on\_response)

send.run()

send.sendmsg.disconnect(slot.get)

## customized widget

QPainter's logical coordinates & device coordinates

QPainter有两个独立的坐标系: 设备坐标系 （用来匹配窗口部件区域中的像素）和逻辑坐标系。

默认情况下，逻辑坐标系和设备坐标系完全对应

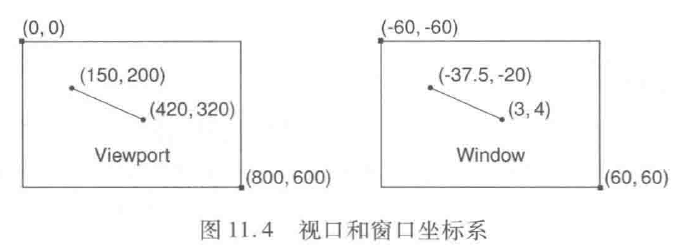
By default the logical and physical coordinate systems coincide, and are equivalent to the paint device's rectangle.

在PyQt术语中，设备坐标系也称为视口viewport, 逻辑坐标系称为窗口window

QPainter painter(this);

painter.setWindow(QRect(-60, -60, 120, 120));

所有的绘制工作都会根据逻辑window坐标系，然后QPainter会自动将逻辑坐标系映射到底层的设备坐标系。



使用窗口window的最大好处在于，可使用逻辑坐标系进行绘制。所有必要的缩放（譬如，当用户调整窗口部件的大小时），将会由PyQt自动完成。对于要绘制文本的自定义窗口部件，要使用物理坐标

Example 见Python Qt GUI快速编程PyQt编程指南 第11单自定义窗口部件

class FractionSlider(QWidget):

def \_\_init\_\_(self, parent=None):

...

self.setFocusPolicy(Qt.WheelFocus) #使用鼠标滚轮时获得焦点

self.setSizePolicy(QSizePolicy(QSizePolicy.MinimumExpanding, QSizePolicy.Fixed))

def sizeHint(self):

return self.minimumSizeHint()

def minimumSize(self):

...

return QSize(...)

def mousePressEvent(self, event):

if event.button() == Qt.LeftButton:

self.moveSlider(event.x())

event.accept()

else:

QWidget.mousePressEvent(self, event)

def mouseMoveEvent(self, event):

self.moveSlider(event.x())

def keyPressEvent(self, event):

change = 0

if event.key() == Qt.Key\_Home: change = -self.max\_value

elif event.key() in (Qt.Key\_Up, Qt.Key\_Right): change = 1

elif event.key() == Qt.Key\_PageUp: change = self.max\_value // 10 + 1

elif event.key() in (Qt.Key\_Down, Qt.Key\_Left): change = -1

elif event.key() == Qt.Key\_PageDown: change = - (self.max\_value // 10 + 1)

elif event.key() == Qt.Key\_End: change = self.max\_value

if change:

value = self.current\_value + change

value = max(0, min(value, self.max\_value))

if value != self.current\_value:

self.current\_value = value

self.emit(SIGNAL("valueChanged(int)"), self.current\_value)

self.update()

event.accept()

else:

QWidget.keyPressEvent(self, event)

def paintEvent(self, event=None):

self.width()

self.current\_value, self.max\_value

...

QPainter

绘制操作在QWidget.paintEvent()中完成，绘制方法必须放在QtGui.QPainter对象的begin（）和end（）之间

PyQt5的绘图API

def paintEvent(self, event):

qp = QPainter()

**qp.begin(self)**

qp.setPen(QColor(168, 34, 3))

qp.setFont(QFont('Decorative', 10))

qp.drawText(event.rect(), Qt.AlignCenter, 'text')

size = self.size()

for i in range(1000):

x = random.randint(1, size().width()-1)

y = random.randint(1, size.height()-1)

qp.drawPoint(x, y)

qp.setPen(QColor(0, 0, 0))

qp.setBrush(QColor(200, 0, 0))

qp.drawRect(10, 15, 90, 60)

…

**qp.end()**

Q & A:

Directory dialog

<https://stackoverflow.com/questions/38746002/pyqt-qfiledialog-directly-browse-to-a-folder?rq=1>

how to get item in QGridLayout?

<https://stackoverflow.com/questions/13020836/qt-getting-column-position-of-clicked-item>

add scroll view ?

<https://stackoverflow.com/questions/41616864/pyqt-expand-grid-in-scroll-area>

gridLayout = QtWidgets.QGridLayout()

scrollAreaWidgetContents = QtWidgets.QWidget()

scrollAreaWidgetContents.setLayout(gridLayout)

scrollArea = QtWidgets.QScrollArea()

scrollArea.setWidgetResizable(True)

scrollArea.setWidget(scrollAreaWidgetContents)

layout = QtWidgets.QHBoxLayout()

layout.addWidget(scrollArea)

replace QLabel with QPushbutton

<https://blog.csdn.net/weixin_30542079/article/details/96321076>

img = QtGui.QImage(r'd:/test.png')

pixmap = QtGui.QPixmap(img)

fitPixmap = pixmap.scaled(64, 64, QtCore.Qt.IgnoreAspectRatio, QtCore.Qt.SmoothTransformation)

btn = QtWidgets.QPushButton()

btn.setIcon(QIcon(fitPixmap))

btn.setIconSize(64, 64)

qApp.setStyleSheet("QPushButton#okButton::hover, click {border: 2px solid white;}")

image viewer

def keyPressEvent(self, keyevent):

""" Capture key to next image, previous image,

on Key Right and key left respectively.

"""

event = keyevent.key()

if event == QtCore.Qt.Key\_Left:

self.nextImage(step=1)

if event == QtCore.Qt.Key\_Right:

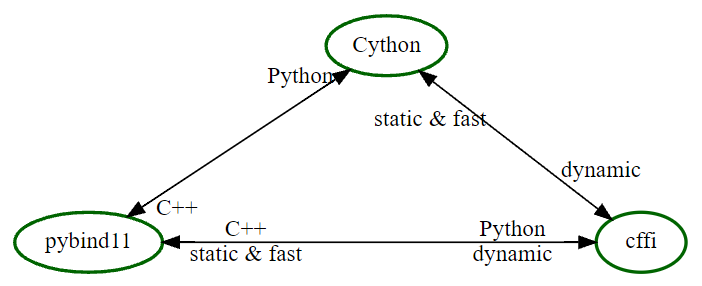
self.nextImage(step=-1)

draw flag

<https://likegeeks.com/pyqt5-drawing-tutorial/>

<https://maicss.gitbooks.io/pyqt5/content/%E7%BB%98%E5%9B%BE.html>

# mix programming: Python <-> C/C++



## Cython

Cython is Python with C data types.

The Cython compiler will convert it into C code which makes equivalent calls to the Python/C API.

Setup.py, which is like a python Makefile

<https://cython.readthedocs.io/en/latest/src/tutorial/cython_tutorial.html>

<https://cython.readthedocs.io/en/latest/src/userguide/index.html>

动态链接库\*.so or \*.pwd

Cython code: \*.pyx

Generated C code

Setup.py

$python setup.py build\_ext -- inplace

(C++ compiler)

C code: \*.h, \*.cpp

Cython compiler

1. **Customized C code: cfib.h, cfib.c**

cfib.h

#ifndef \_\_CFIB\_H\_

#define \_\_CFIB\_H\_

double cfib(int n);

#endif

cfib.c

#include “cfib.h”

double cfib(int n) { … }

**Build a Cython extension: wrap\_fib.pyx**

# declare interface

cdef extern from “cfib.h”:

double cfib(int n)

# create Cython wrapper

def fib(n):

return cfib(n)

**Compilation and Importing: setup.py -> wrap\_fib.so or wrap\_fib.pyd**

setup.py

from setuptools import setup, Extension

from Cython.Build import cythonize

ext = Extension(name=’wrap\_fib’, sources=[‘cfib.c’, ‘wrap\_fib.pyx’])

setup (ext\_modules = cythonize(ext))

$ python setup.py build\_ext - - inplace

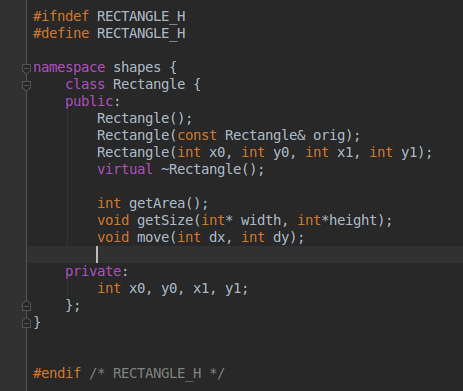
生成wrap\_fib.so in unix or wrap\_fib.pyd in window

\*.py call wrap\_fib.so

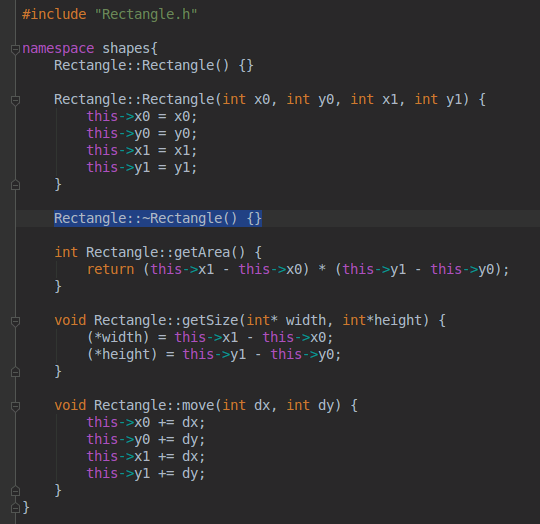
from wrap\_fib import fib

1. **Customized C++ code: Rectangle.h, Rectangle.cpp**

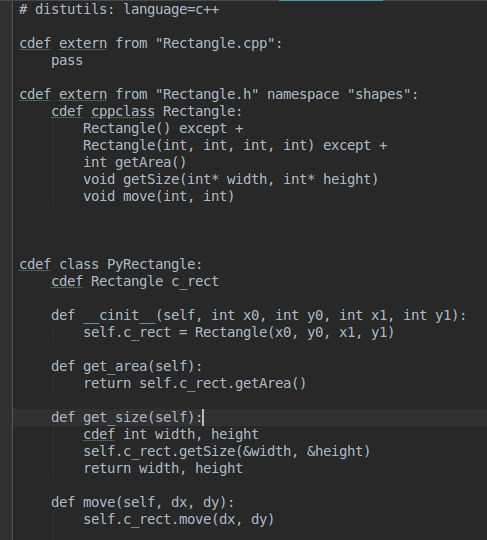
Rectangle.h



Rectangle.cpp

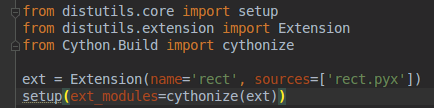


**Build a Cython extension: rect.pyx**



**Compilation and Importing: setup.py -> rect.so or rect.pyd**

setup.py



$ python setup.py build\_ext - - inplace

生成rect.so in unix or rect.pyd in window

\*.py call rect.so

from rect import PyRectangle

1. **STL C++ code**

wrapper.pyx

# distutils: language=c++

from libcpp.algorithm cimport sort

from libcpp.vector cimport vector

# 可以直接调用cpython封装好的STL

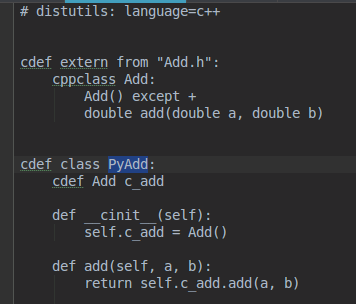
def cppsort(int[:] x):

sort(&x[0], &x[-1]+1)

1. Library

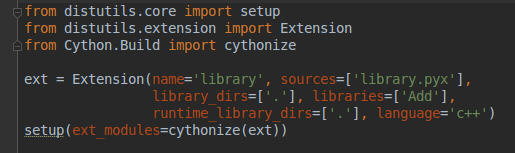
假定你已实现动态库Add.h, libAdd.so (Add.dll)

**Build a Cython extension: library.pyx**



**Compilation and Importing: setup.py -> library.so or library.pyd**

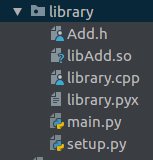
setup.py



$ python setup.py build\_ext - - inplace

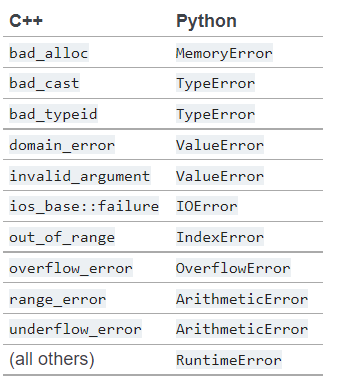
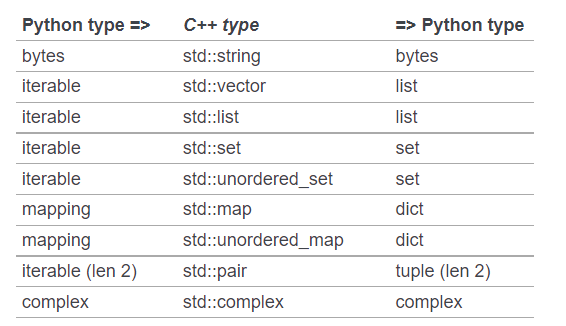
生成library.so in unix or library.pyd in window

main.py调用libAdd.so

Cython语法

Most of the containers of the C++ Standard Library have been declared in pxd files located in /Cython/Includes/libcpp



## Boost.Python

expose C++ classes functions and objects to Python, and vice-versa

demo

src

classes.h/cpp

classes.py

CMakeLists.txt

CMakeLists.txt

**src/classes.h**

#include <string>

class RealWorld {

public:

RealWorld(std::string n, char sex) : name(n), sex(‘m’), age(0.0) {};

void Welcome();

void SetAge(int age);

int GetAge();

std::string GetName();

std::string GetSex();

private:

char sex;

int age;

std::string name;

}

**src/classes.cpp**

#include <iostream>

#include <boost/python.hpp>

#include “classes.hpp”

namespace python = boost::python;

void RealWorld::Welcome() {…}

int RealWorld::GetAge() {…}

void RealWorld::SetAge(int value) { …}

std::string RealWorld::GetName() {…}

char RealWorld::GetSex() {…}

BOOST\_PYTHON\_MODULE(classes) { //模块

Python::class\_<RealWorld>(“RealWorld”, python::init<std::string, char>()) //类

// Expose functions

.def(“Welcome”, &RealWorld::Welcome)

.def(“GetAge”, &RealWorld::GetAge)

.def(“SetAge”, &RealWorld::SetAge, python::args(“value”))

.def(“GetName”, &RealWorld::GetName)

.def(“GetSex”, &RealWorld::GetSex)

// Expose member

.add\_property(“age”, &RealWorld::GetAge, &RealWorld::SetAge)

.add\_property(“name”, &RealWorld::GetName)

.add\_property(“sex”, &RealWorld::GetSex)

}

**src/CMakeLists.txt**

set(MODULE\_NAME classes)

include\_directories($(CMAKE\_SOURCE\_DIR))

add\_libraries(${MODULE\_NAME} SHARED classes.cpp)

if (UNIX) set\_target\_properties(${MODULE\_NAME} PROPERTIES PREFIX “”)

elseif (WIN32) set\_target\_properties(${MODULE\_NAME} PROPERTIES SUFFIX “.pyd”)

endif()

target\_link\_libraries(${MODULE\_NAME} ${Boost\_LIBRARIES} ${PYTHON\_LIBRARIES})

**src/classes.py**

import classes

t1 = classes.RealWorld(“hi”, “m”)

t1.Welcome()

…

**CMakeLists.txt**

ADD\_SUBDIRECTORY(src)

// build and run

demo$cd build

build$ cmake ..

build$ make

demo/build$ cd lib

demo/build/lib$ cp ../../src/classes.py .

demo/build/lib$ python classes.py