目录

让你人生不苦短的编程语言：Python

- 最新版本：`v1.1`
- 更新时间：`20210802`

简介

介绍为何Python编程语言可以让你人生不苦短。包括Python语言的历史概述、能用Python做什么、关于Python的一些常见谬误；介绍Python基础开发，包括选择Python2还是Python3、常见的Python的编辑器和IDE，比如系统自带的文本编辑器、VSCode、PyCharm等；以及Python的语法，包括基础通用语法和Python特有语法，以及Python的内置函数；接着介绍Python的某些专题，比如字符串、正则re、迭代器、dict字典、logging日志、copy拷贝、import导入、类和对象、site-packages、自带IDE即IDLE、何测试、多线程、历史版本等；接着介绍Python的各个技术领域的开发，包括多版本和虚拟环境、表格处理、Web框架、数据科学和人工智能、网络、图像处理、爬虫、GUI图形界面、数据库、打包、配置管理、项目部署、服务监控等；以及介绍其他Python相关内容，比如PEP、常用Python库、一些有趣的库、开发iOS或Mac程序、SSH等；最后附上Python相关教程。

源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

Gitbook源码

- crifan/make_life_better_python: 让你人生不苦短的编程语言：Python

如何使用此Gitbook源码去生成发布为电子书

详见：crifan/gitbook_template: demo how to use crifan gitbook template and demo

在线浏览

- 让你人生不苦短的编程语言：Python book.crifan.com
- 让你人生不苦短的编程语言：Python crifan.github.io

离线下载阅读

- 让你人生不苦短的编程语言：Python PDF
- 让你人生不苦短的编程语言：Python ePub
- 让你人生不苦短的编程语言：Python Mobi

版权说明

此电子书教程的全部内容，如无特别说明，均为本人原创和整理。其中部分内容参考自网络，均已备注了出处。如有发现侵犯您版权，请通过邮箱联系我 `admin 艾特 crifan.com`，我会尽快删除。谢谢合作。

鸣谢

感谢我的老婆**陈雪**的包容理解和悉心照料，才使得我 `crifan` 有更多精力去专注技术专研和整理归纳出这些电子书和技术教程，特此鸣谢。

更多其他电子书

本人 `crifan` 还写了其他 `100+` 本电子书教程，感兴趣可移步至：

crifan/crifan_ebook_readme: Crifan的电子书的使用说明

Python概述

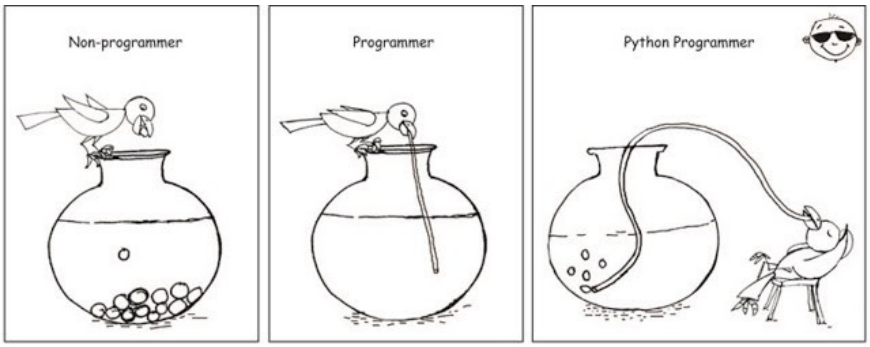
现有编程语言众多，其中有个叫 `Python`

不同的人对其有各种不同的评价，其中让人最为印象深刻和精辟的一个评价是：

人生苦短，我用Python

意思就是：相对其他语言来说，用Python搞开发，可以让你人生不那么苦短

借用某图来说明 = 一图胜千言：



形象的表现出：

如果会了Python，你可以在编程(工作)期间

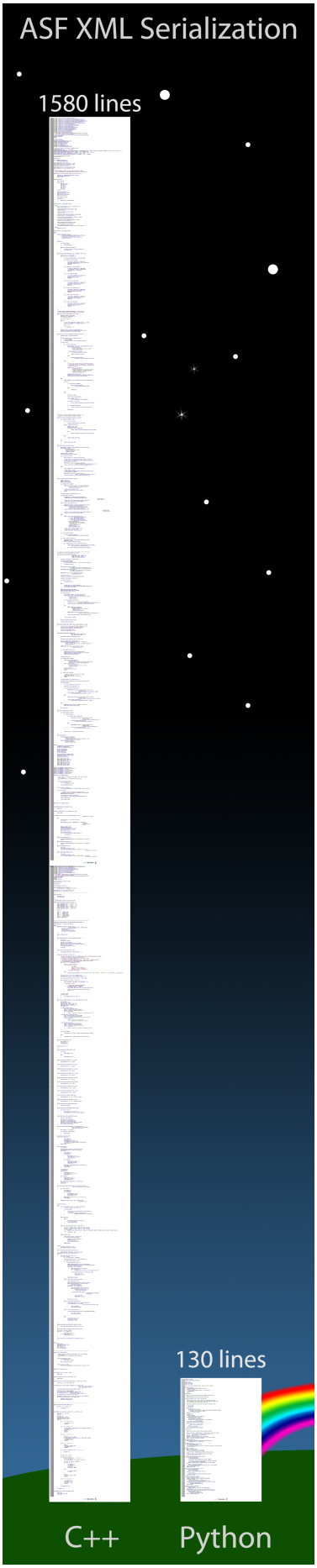
高效的、方便的实现各种工具和系统，从而

-> **让（打工）人生不（那么）苦短**

-> **用更少的代码，更好的实现更多的功能**

举例：实现同样功能的 `C++` 和 `Python` 的代码量的对比

此处通过，实现同样的功能， `ASF XML Serialization` ，所需要的 `C++` 代码和 `Python` 代码做个对比：


ASF XML Serialization
1580 lines
130 lines
C++
Python

就更能直观的体会到，什么叫做：

人生苦短，我用Python

即：用别的语言搞技术开发，有多么**苦逼**了。

Python历史

此处概述Python的大体发展历史

- **Python 1.0时代：起源与诞生**
  - 谁开发了Python：`Guido van Rossum` =简称 `Guido`
    - -> `Guido` 是Python语言之父
    - 
    - 相关背景
      - 1982年从阿姆斯特丹大学获得了数学和计算机硕士双学位，
      - 期间他接触了很多的语言，包括 `Pascal` ， `C` ， `Fortran` 等
      - 很多编程语言更侧重于像计算机一起思考
        - 而不是更像人一样去思考
          - 所以作为人类，去写和用各种编程语言，就很麻烦，费解
      - 非常欣赏shell
        - shell简单易编程的特性，让程序员更加专注于设计和逻辑本身
          - 但shell也不算是一门 语言
            - shell本质上是一个功能的调用
      - 希望：一种 （像 `shell` 一样）简单易用，且（像 `C` 一样）功能又强大的语言
      - 现实：不存在
      - 所以：自己开发一个
    - 开发过程
      - 1989年的圣诞节，Guido开始编写Python语言的编译器
      - Python这个名字来源于他喜欢的电视剧 `Monty Python's Flying Circus` ，而不是表面意义上的“蟒蛇”
      - 他希望这个新的语言，能符合他的理想：介于C和shell之间，功能全面、易学、易用又可拓展
    - 第一版
      - 1991年，第一个Python编译器诞生，这标志着Python的第一个版本正式诞生

        - 它基于C语言，并具备了基础的类、函数、异常处理等功能特性，同时具备可扩展性。Python语法很多来自C，但又受到ABC语言的强烈影响
            - 例如来源于ABC语言强制缩进的规定本身可以让Python容易读，但如果缩进出错却会影响编译和执行
    - 后续发展
        - Python本身不以性能为重，但当确实需要考虑性能时，Python程序员却可以深入底层来编写C程序，并编译为.so文件引入到Python中使用
        - 所以后期被很多人称为：**胶水语言**
            - 可以做多个语言中间的**胶水**
        - Python语言的魅力在于让程序员可以花更多的时间用于思考程序的逻辑，而不是具体的实现细节，这一特性也得到Guido同事的欢迎。
        - 他们在反馈使用意见的同时也参与到Python的改进中来，因此最初Guido和一些同事构成了Python的核心团队，当然，核心决策者还是Guido本人。
        - 随后，Python的使用拓展到研究所之外，并吸引了越来越多的程序员。
        - 但是，最初Python的使用非常小众，因为在那个计算机资源非常有限的年代，大家都倾向于最大化榨取计算机资源并提升运算效率，而Python显然不是为此而生。
- **Python 2.0时代：崛起**
    - 最初发布时，Python在设计层面存在一些缺陷，例如以满足跨语言、跨平台进行文本转换、处理的要求的Unicode字符编码标准在1994年才正式公布，所以一直以来Python 2及之前的版本对Unicode的支持并不完全。相信大家在使用Python 2版本处理中文时都遇到过各种问题。
    - 2000年发布的Python 2.0标志着Python的框架基本确定。重要框架方向包括：
        - **简单明确**
            - 在设计Python语言时，开发者倾向于选择没有或者很少有歧义的语法。由于这种设计观念的差异，Python源代码通常被认为比Perl具备更好的可读性，并且能够支撑大规模的软件开发。
        - **面向对象**
            - 任何Python的元素都可以视为对象，包括数据类型、类、函数、实例化元素等，完全支持继承、重载关系，这有益于增强代码的复用性。
        - **动态类型**
            - 任何对象的数据类型都无需提前定义，拿来即用。即使在之前已经预先定义，后期也可随时修改。
        - **胶水特性**
            - Python本身被设计为可扩充的，并非所有的特性和功能都集成到语言核心。Python提供了丰富的API和工具，以便程序员能够轻松地使用C、C++、Cython来编写扩充模块。例如在Google对于Google Engine使用C++编写性能要求极高的部分，然后用Python或Java/Go调用相应的模块。
        - **可嵌入**

        - 你可以把Python的功能嵌入到C/C++程序中，从而实现Python功能在其他语言中的功能实现。
    - **生态系统**
        - Python有强大的标准库，同时支持第三方库和包的扩展应用，甚至可以自定义任何库和包。Pypi是其第三方库的仓库，在这里你几乎可以找到任何领域内的功能库。
    - **解释器机制**
        - Python支持多种解释器，例如
            - `CPython`：官方版本，基于C语言开发，也是使用最广的Python解释器
            - `IPython`：基于CPython之上的一个交互式解释器
            - `PyPy`：一个追求执行速度的Python解释器，采用JIT技术对Python代码进行动态编译
            - `Jython`：运行在Java平台上的Python解释器，可以直接把Python代码编译成Java字节码执行
            - `IronPython`：和Jython类似，只不过运行在微软的`.Net`平台上
- 1965年，戈登·摩尔提出了著名的摩尔定律，其内容为：当价格不变时，集成电路上可容纳的元器件的数目，约每隔18-24个月便会增加一倍，性能也将提升一倍。在随后超过半个世纪的时间里，个人计算机的发展日新月异，已经由资源不足向资源过剩转变。这客观上为Python的应用提供了基础条件——只有在资源过剩的条件下，程序员才不会过度关注榨取性能
- 随着Python自身功能的完善以及生态系统的扩展，Python在Web开发、网络爬虫、数据分析与数据挖掘、人工智能等应用方面逐渐崭露头角：
    - **Django和Flask引领的WEB开发模式**
        - 2004年，目前最流行的WEB框架Django诞生。2010年，另一个流行的轻量级WEB框架Flask诞生。Django是一个WEB解决方案“全家桶”，其功能大而全，包含了几乎所有WEB开发相关的组件和功能，它可以大大节省开发者在基础组件、选型、适配等方面的时间和精力；而Flask只包含基本的配置，默认依赖于两个外部库也可以自由替换，给开发者提供最大的自主空间。这两类完全相反方向上的WEB开发模式，几乎可以为所有开发者提供了很好的选型参照物：无论开发者想要一站式还是最大化自主解决方案，Python都能满足
        - 此后，以豆瓣、春雨医生、知乎、Dropbox、YouTube、CIA（美国中情局）等为代表的企业和机构都基于Python做网站开发，预示着Python应用到WEB开发领域逐渐成为一种新兴趋势
    - **人人都能胜任的网络爬虫**
        - Python自带的标准库中，urllib、urllib2、requests库对于简单网页的抓取实现非常简单，即使在面对海量数据抓取需求时，第三方库Scrapy也能应对自如；再配合正则表达式库re、网页代码解析BeautifulSoup、html和xml解析库lxml、多线程库threading等特性，使得Python在应用到网络爬虫任务上时，只需要很少的开发量便能迅速完成任务。基于Python简单易学的特性，几乎人人都能开发网络爬虫
    - **比shell更好用的自动化运维工具**

        - Python是跨语言和平台的，几乎所有Linux系统和MAC系统都自带Python库，Windows系统也可以自定义安装。Python默认的os、sys等库可实现与操作系统的交互和执行功能，更重要的是Python还能直接执行系统终端命令。因此，使用Python编写的系统运维和管理脚本在可读性、性能、代码重用度、扩展性几方面都优于普通的shell脚本，在自动化运维方面应用广泛
    - **数据分析与科学计算三剑客**
        - 2008年发布的Numpy、scipy和2009年发布的pandas是数据分析与科学计算的三剑客。
            - `NumPy` = `Numeric Python`
                - Python科学计算的基础工具包，也是Python做数据计算的关键库之一，同时又是很多第三方库的依赖库
            - `Scipy` = `Scientific Computing Tools for Python`
                - 一组专门解决科学和工程计算不同场景的主题工具包，它提供的主要功能侧重于数学、函数等，例如积分和微分方程求解
            - `Pandas` = `Python Data Analysis Library`
                - 一个用于Python数据分析的库，它的主要作用是进行数据分析和预处理。
                - Pandas提供用于进行结构化数据分析的二维表格型数据结构DataFrame，类似于R中的数据框，能提供类似于数据库中的切片、切块、聚合、选择子集等精细化操作，为数据分析提供便捷。另外，Pandas还提供了时间序列的功能，用于金融行业的数据分析
    - 除此之外，很多大型公司也都在使用Python完成不同类型的其他工作，其中不乏世界知名公司，如国外的Google、Facebook、NASA 、雅虎、YouTube等，国内的网易、腾讯、搜狐、金山等。例如谷歌在Google Groups、Gmail、Google Maps等项目中将Python用作网络应用的后端；在Google Cloud Platform中的Google Cloud Storage本地部署环境中，gsutil也在Python 2基础上开发和应用。
- **后Python2与Python3时代，AI让Python大放异彩**
    - 2008年12月，Python 3发布。Python 3相对于Python 2的早期版本（主要是Python2.6之前）是一个较大的升级，它在设计的时候没有考虑向下兼容，所以很多早期版本的Python程序无法在Python 3上运行。为了照顾早期的版本，推出过渡版本2.6——基本使用了Python 2.x的语法和库，同时考虑了向Python 3.0的迁移，允许使用部分Python 3.0的语法与函数。同时，Python还提供了Python 2到Python 3的Python文件转换功能，以帮助开发者升级
    - 2010年7月发布了Python 2.x系列的最后一个版本，主版本号为2.7。大量Python 3的特性被反向迁移到了Python 2.7，2.7版本比2.6版本进步非常多，同时拥有大量Python 3中的特性和库，并且照顾了原有的Python开发人群。Python2.7也是当前绝大多数Linux操作系统最新版本的默认Python版本。
    - 从2008年开始，Python 2与Python 3是并存发展的。但在2018年3月，Guido在邮件列表上宣布Python 2.7将于2020年1月1日终止支持，这意味着之后Python 2将不再被统一维护，与之对应的是主流第三方库也不会再提供针对Python 2版本的开发支持。Python 2的时代即将过去。

- 这一时期，Python继续以其独特魅力吸引更多的开发者加入，但真正让Python大放异彩的却是AI（人工智能）的爆发。
- AI并不是一个新生事物，而是从20世纪50年代就开始出现，随后经过了大概20年的黄金时期，又分别在20世纪70年代和90年代两次进入寒冬期。从2006年开始，神经网络、深度学习的出现，让AI进入爆发期。
- 在AI领域，Python拥有很多相关库和框架。其中最著名的是：
  - **sklearn**：一个老牌机器学习库，其neural_network库可用来做神经网络训练。
  - **PyTorch**：由Facebook于2016年发布，它基于曾经非常流行的Torch框架而来，为深度学习的普及迈出了重要一步，到目前为止它已经是人们用来做学术研究的首选方案
  - **TensorFlow**：谷歌于2015年研发的第二代人工智能学习系统。借助谷歌的强大号召力以及在人工智能领域的技术实力，它已经成为目前企业真实生产环境中最流行的开源AI框架。更重要的是，它也是第一个（应该也是唯一一个）经过真实大规模生产环境（Google）检验过的框架
- 在互联网领域，Facebook和Google都是全球IT企业的标杆，具备行业领导力和风向指示意义。他们基于Python开发的AI库（PyTorch和TensorFlow）已经成为目前最流行的AI库，而且“到底选择PyTorch还是Tensorflow”仍然是一个具有争议性的话题
- 在AI时代，主要应用场景包括：
  - **计算机视觉=CV**：通过特定的图片模式训练，让计算机理解图像中的物体甚至内容。在这一领域我们熟悉的场景包括图像识别、目标识别和跟踪。例如人脸识别便是图像识别的典型领域，广泛应用到企业员工考勤、门店客户识别、机场等公共领域反恐识别等。2011年，吴恩达创立的谷歌大脑项目，能够在没有任何先验知识的情况下，仅仅通过观看无标注视频学习到识别高级别的概念就能知道哪个是猫。
  - **(自动)语音识别=ASR**：该过程是计算机将人类的自然语言识别并转换为文字的过程，广泛应用工业、家电、通信、汽车电子、医疗、家庭服务、消费电子产品等各个领域。身边熟悉的场景例如通过语言对导航、APP、车载设备等做指令输入，以及电信客服系统中的语音业务查询和办理
  - **自然语言理解=NLP**：自然语言理解是一类任务的总称，而并非是单一任务。它旨在让计算机理解人类的语言所表达的表层和深层含义。目前场见的应用场景包括自动问答系统、机器翻译、信息检索和过滤、信息抽取等
  - **个性化推荐**：个性化推荐是一个相对成熟的领域，但基于深度学习和神经网络，可以将大量的复杂、抽象特征的数据预处理工作最大程度的简化，甚至可以将海量特征经过简单处理后便直接丢到模型中便能获得比例理想的效果。
  - **游戏和竞技**：在该领域，很多科技公司用经过训练后的AI与人类进行对弈。早在20世纪90年代，由IBM开发的“深蓝”与卡斯帕罗夫的世纪之战已经引起了世界的轰动；在2017年AlphaGo又击败排名世界围棋冠军柯洁，再一次让世人感受到AI的强大威力
- 在不同的领域，Python都能扮演非常重要的角色，因此，在国外的各大榜单中，Python都已经成为最受欢迎的语言（或至少是之一）。不只在商业领域流行，国内很多地区和教育机构正将Python纳入教材之中。比如

Python进入山东小学六年级的教材，浙江信息技术教材将放弃VB，改用Python 语言，Python列入全国计算机二级等级考试大纲等。

- **Python的未来发展**
  - 在Python发展过程中，Guido一直是核心人物，甚至被称为 **终身仁慈独裁者=** `Benevolent Dictator For Life = BDFL`，但在2018年经历了退出管理层风波之后，他又在2019年以五大指导委员之一的身份重回决策层。这为Python迎来了新的治理方案：指导委员会模式。这种模式意味着Python的未来将从Guido一人决定变为5人决定，虽然比很多开源语言仍然有民主化空间（例如PHP的改进由社区投票决定），但也算是一种从专制到民主的进步
  - 有关Python的每个提升计划，都是在PEP（Python Enhancement Proposal）列表中——每个版本新特性和变化都通过PEP提案经过社区决策层讨论、投票决议，最终才有我们看到的功能。
  - 目前，Python的最新稳定的主版本是3.7，Python 3.8也已经有了预览版，大概在2023年左右Python 4便会问世。在之后的时间里，Python会如何发展？我们可以从Python软件基金会的董事会成员、CPython的核心开发人员Nick Coghlan的信息中略知一二：
    - **首先**，Python的PEP流程和制度没有任何变化，通过增加新模块和功能来增强的基础能力。随着Python 2在2020年不再维护，社区在Python 3的资源和投入会相应增加
    - **其次**，不同解释器的实现和功能扩展还将继续增强，方向包括PyPy关于JIT编译器生成和软件事务内存的尝试，以及科学和数据分析社区，对面向数组编程的探索等
    - **再次**，嵌入式应用的增强，核心是与其他虚拟机运行时（如JVM和CLR）的集成和改进，尤其是在教育领域取得的进展，可能会让Python作为更受欢迎的嵌入式脚本语言，在更大的应用程序中运行
    - **最后**，对于为了兼容和维持Python 2的部分功能而存在于Python 3中的原有代码，在后续版本中应该会逐步优化甚至去掉。而对于其他更改，则会根据情况弃用、提出警告、逐步替代以及保留。

Python典型用途

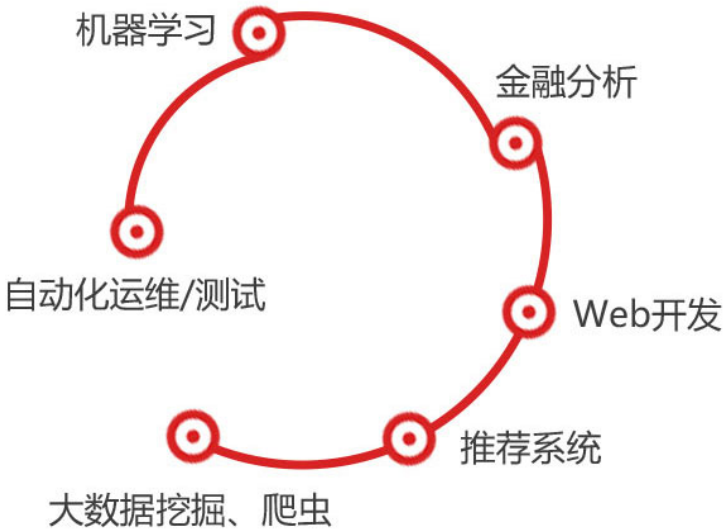
Python语言可以有很多用途。

此处贴出一些统计数据，供了解大概情况：

Python相关工作、职业、公司

- 在用Python的企业
  - 

- Python就业=应用领域



- 典型Python职位 = 工作方向
  - Python 全栈工程师
  - Python Web开发工程师
  - Python 爬虫工程师
  - Python 数据分析师
  - Python 数据挖掘师
  - Python 机器学习工程师
  - Python 数据处理工程师
  - Python 推荐系统工程师
  - Python 推荐系统架构师
- 计算机领域=业界 的一些Python应用

    - 电信基础设施 (Twilio)
    - 支付系统 (PayPal, Balanced Payments)
    - 神经科学和心理学 (许多, 许多, 例子)
    - 数值分析和工程 (numpy, numba, 以及 更多其它)
    - 动画(LucasArts, Disney, Dreamworks)
    - 游戏后台 (Eve Online, Second Life, Battlefield, 以及 其它很多)
    - Email基础设施 (Mailman, Mailgun)
    - 媒体存储和处理 (YouTube, Instagram, Dropbox)
    - 操作和系统管理 (Rackspace, OpenStack)
    - 自然语言处理(NLTK)
    - 机器学习和计算机版本 (scikit-learn, Orange, SimpleCV)
    - 安全性和渗透性测试 (很多很多 以及 eBay/PayPal)
    - 大数据 (Disco, Hadoop support)
    - 如理 (Calendar Server, 它 驱动了 Apple iCal)
    - 搜索系统 (ITA, Ultraseek, 还有 Google)
    - Internet 基础设施 (DNS) (BIND 10)

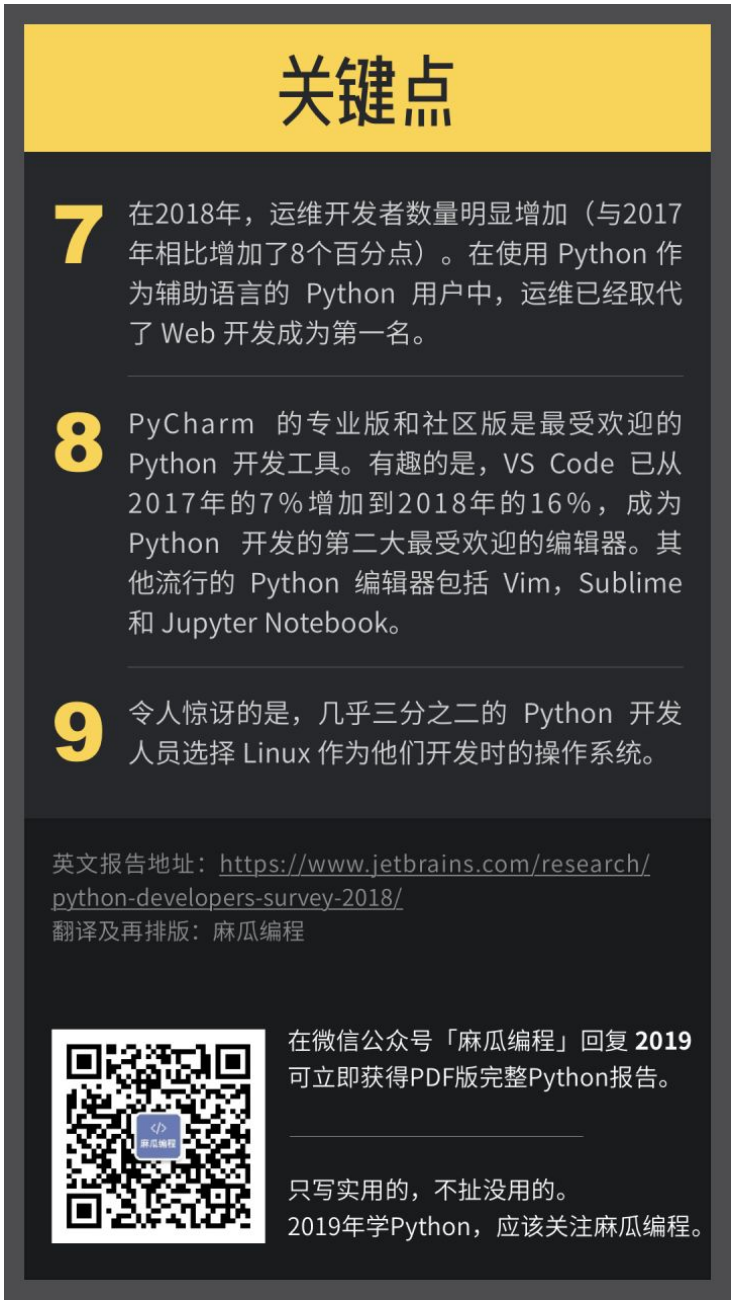
更多Python的应用，可参考官网的列表：Python Success Stories | Python.org

Python的使用

参考一些报告，有些结论供参考：

- Jetbrains的2018年的Python报告
    - 用Python语言的人
        - **84%**是用作主要语言的
        - 除了Python，还用
            - `JavaScript`， `HTML` / `CSS`， `Bash` / `Shell`， `SQL` 等
        - **60%**用作工作（和个人）方面
        - 用来做什么
            - `数据分析`
            - `Web开发`
            - `运维/系统管理/自动脚本`
            - `机器学习`
            - `网页解析和爬虫`
            - `软件测试/自动化测试`
            - `教学目的`
            - `软件原型`
        - Python版本：
            - **84%**都用 `Python3` 了
                - `Web开发`， `数据分析`， `运维` 中用Python3最多
            - **16%**还用Python2
    - 关键点


关键点
1
Python 3 的使用率正在快速增长，并且已经达到84%，而 Python 2 仅被16%的 Python 用户用作主要的解释器。在2017年 Python 3 使用比率是75%，而在2018年 Python 3 变得更受欢迎了。
2
Python 用于数据分析比用于 Web 开发更广泛，从2017年的50%增长到2018年的58%。
3
使用 Python 作为主语言的开发人员中，有一半也使用 JavaScript。 Python也经常与HTML / CSS，Bash / Shell，SQL，C / C ++和Java 一起使用。
4
Flask 和 Django 是 Web 开发人员中最受欢迎的框架，拥有相同的份额（每个约45%），他们远远超越了其他 Python Web 框架。
5
NumPy，Pandas，Matplotlib 和 SciPy 是最受欢迎的数据科学框架和库。机器学习专用的库如SciKit-Learn，TensorFlow，Keras等也很受欢迎。
6
AWS 是 Python 开发人员最受欢迎的云平台，其次是 Google Cloud Platform，Heroku，DigitalOcean 和 Microsoft Azure。



关于Python的谬误

关于Python有些常见的谬误。

现整理如下，供知悉。

Python速度慢

首先要搞清楚一个重要区别: `Python` 是一门**编程语言**，而不是**运行时环境**

Python有几个（具体）实现 = 运行时:

- CPython 是参考（默认的）实现, 且也是广泛发布和使用的实现
- Jython 是Python用于 `JVM` 的是一个成熟的实现
- IronPython 是 `Microsoft` 针对其自家的通用语言运行时——又名 `.NET` 实现的Python
- PyPy 是一个正在日趋成熟的Python实现，拥有 `JIT` 编译，增量垃圾收集诸多先进的特性

每一个运行时都有其自己的性能特点, 而且它们本身也不慢。

这里更重要的地方在于

- 不能错误地用性能指标去评价**Python编程语言**
  - 而应该去用性能指标去评价一个**Python的运行时**
    - 且最好是针对一个特定的使用场景

清楚了上述逻辑后，下面给出一些例子来说明，其实Python（的运行时，在对应应用场景下）性能其实很强:

- 把 NumPy 用作 Intel 的 MKL SIMD接口
- PyPy的 `JIT` 编译能达到比C还快的性能
- Disqus能在同样的100个盒子上容纳两亿五千万到5亿用户

Python基础开发

Python2还是Python3

在（2020年）之前，入门Python时，还会遇到一个问题：

到底是选择（生态相对较广的）`Python 2`，还是选择（版本更新的）`Python 3`？

之前的回复一般是：新手可以考虑`Python 2`，稍微熟悉后换`Python 3`

而现在（2020年之后），官网（2020年之后就）早已放弃`Python 2`。

所以此问题已不存在，直接选择`Python 3`入手学习和后续开发，即可。

另外补充几句：

- 现在Python的生态中，还是有（一些、或者说不少）`Python 2`的库的
  - 但只要是活跃的、有生命力的库，会（早晚）尽快换到（支持）`Python 3`的
- `Python 3`的**性能**总体上比`Python 2`更好
  - 严格的说：`Python 3`的具体实现，往往比之前`Python 2`的实现，总体上性能更好
    - 大概水平：`Python 3`比`Python 2`快大概**1.2倍**左右
  - 细节的说：对于Python的解释器版本，`CPython`，`PyPy`：即时（JIT）编译器等来说
    - `PyPy`比`CPython`更快

Python的IDE

Python开发，有很多编辑器和IDE可以用。

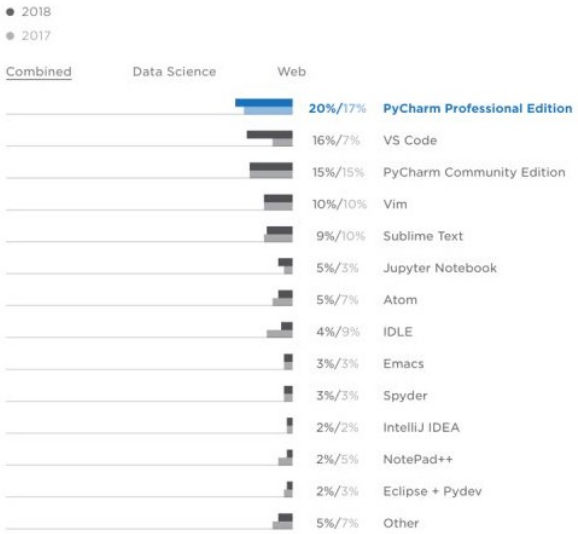
关于Python的IDE选择，总的意见是：

- 新手：系统自带**文本编辑器**
  - 原因：先搞懂基本的Python开发逻辑和流程，再换用更好更高级的IDE，利于循序渐进的学习
- 老手：优先推荐 VSCode，其次推荐 PyCharm
  - 原因：充分利用 `VSCode` 和 `PyCharm` 的功能，打造顺手的工具，提高开发效率

附带：

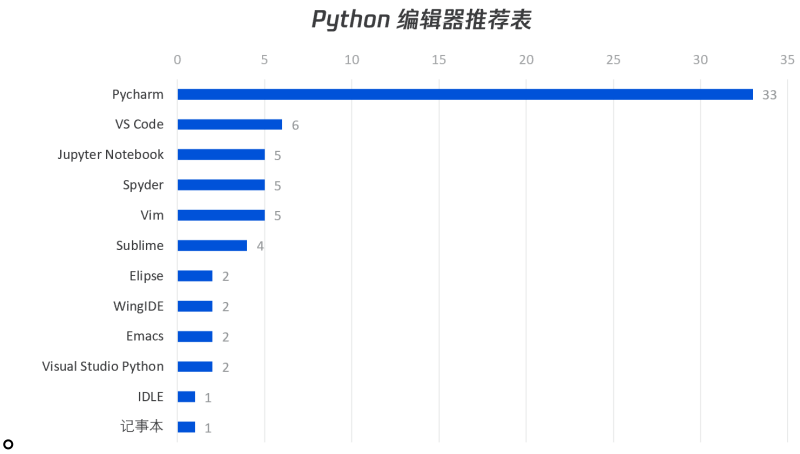
- 做数据分析的：建议用 `Spyder`
  - 且可以考虑安装 `anaconda`，其中自带了 `Spyder`
- 以前也试过其他的：`PyScripter`
  - 【记录】使用Python的IDE：PyScripter – 在路上
- 相关：
  - 编辑器和IDE总结
    - 【整理】各种Python的IDE(集成开发环境)的总结和对比 – 在路上
      - 4.1.3.2. 目前常见的一些Python的IDE - - python初级教程：入门详解
- 相关Python的IDE统计，供参考：

编辑器和 IDE



PyCharm 是最受欢迎的Python开发工具，PyCharm 专业版和社区版的合计份额为35%。有趣的是，VS Code 从2017年的7%上升到2018年的16%，成为第二个最受欢迎的 Python 开发编辑器。 很可能是因为 VS Code 的快速增长，许多其他编辑器的用户份额减少了。

Web 开发人员与数据科学家的偏好略有不同。 他们比数据科学家更喜欢PyCharm，VS Code，Vim 和 Sublime。而许多数据科学家更喜欢 Jupyter Notebook 作为他们的主要工具。



文本编辑器

VSCode

详见独立教程：

调试Python · 史上最好用的编辑器：VSCode

PyCharm

详见独立教程：

最智能的Python的IDE：PyCharm

Python语法

编程语言通用语法

- 快速了解Python的基本语法
  - 一图搞懂Python基本语法

  - 源码：py3SyntaxInOnePic.py
  - 原始(Snagit)文件：一图搞懂Python基本语法_20210410.snagproj

Python特有语法

代码缩进 表示 逻辑语义

Python中，和很多其他编程语言，在（空格、Tab制表符等）缩进上，不一样：

- 其他编程语言：缩进，往往是没有语法含义的，只是为了美观和易读
- `Python` ： 缩进，是有意义的，决定了代码的逻辑层次
  - 不同缩进层次，对应了不同的代码块
  - 别人评论
    - 缩进和空白是有意义的 -》 表示代码块
      - 这个叫做：offside rule
        - 注：
          - Off-side rule - Wikipedia
            - 类似于足球中的越位
              - 把本来没有意义的空白（和缩进），用于代码的缩进，有逻辑上的含义
                - -》翻译成汉语，可以说成是：过犹不及

TODO：

- 把之前的单独文章整理过来
  - 【教程】详解Python中代码缩进（Indent）：影响代码的内在逻辑关系和执行结果 – 在路上

双星号 = 两个星号 = `**`

Python中的 `**` 是表示（被用来）展开传入函数的参数的 = 展开（函数参数）字段

典型场景：

想要给一个函数传递多个参数，但是又不确定传递几个

如果用： `if else` 写法，会很繁琐

此时可以用： `dict` 字典保存要传递的参数，在计算出要传递的参数后，最终用 `**paramDict` 传递参数进入

举例：同步印象笔记的笔记时的多参数传递

普通写法 = `if else` 写法：

```
    # 2. Sync to Evernote
    if isNewRes:
      if needUpdateAttributes:
        respNote = gEvernote.syncNote(
          noteGuid=noteDetail.guid,
          noteTitle=noteDetail.title,
          newContent=noteDetail.content,
          newResList=noteDetail.resources,
          newAttributes=noteDetail.attributes
        )
      else:
        respNote = gEvernote.syncNote(
          noteGuid=noteDetail.guid,
          noteTitle=noteDetail.title,
          newContent=noteDetail.content,
          newResList=noteDetail.resources
        )
    else:
      if needUpdateAttributes:
        respNote = gEvernote.syncNote(
          noteGuid=noteDetail.guid,
          noteTitle=noteDetail.title,
          newContent=noteDetail.content,
          newAttributes=noteDetail.attributes
        )
      else:
        respNote = gEvernote.syncNote(
          noteGuid=noteDetail.guid,
          noteTitle=noteDetail.title,
          newContent=noteDetail.content
        )
```

注：且如果参数更多，代码看起来会更乱

而双星号写法，就很简单，易读，易扩展：

```
    # 2. Sync to Evernote
    syncParamDict = {
        # mandatory
        "noteGuid": noteDetail.guid,
        "noteTitle": noteDetail.title,
        # optional
        "newContent": noteDetail.content,
    }

    if isNewRes:
        syncParamDict["newResList"] = noteDetail.resources

    if needUpdateAttributes:
        syncParamDict["newAttributes"] = noteDetail.attributes

    respNote = gEvernote.syncNote(**syncParamDict)
```

注意事项

`Pvthon version < 3.5 not allow keyword argument after **exception`

`Python 3.5` 之前的版本，比如 `Python 3.4` 中，如下写法：

```
self.connection = pymysql.connect(**self.config, cursorclass=pymysql.cursors.D
```

会报错：

```
    self.connection = pymysql.connect(**self.config, cursorclass=pymysql.cursor
                                                     ^
SyntaxError: invalid syntax
```

原因：此处的Python版本小于 `3.5` ，不支持 `**someDictOrTuple` 之后，再跟着 `其他参数` 的写法

解决办法：代码改为：

```
self.config["cursorclass"] = pymysql.cursors.DictCursor
self.connection = pymysql.connect(**self.config)
```

即可。

`with`

详见：

【整理】python中的with的含义和用法

魔术方法 特殊方法

详见：

【整理】Python 特殊方法 魔术方法

yield

详见：

【已解决】Python中直接return数组和yield的区别

Python内置函数

Python中有很多内置的、自带的函数，功能很强大，很好用。现整理如下。

hasattr

举例说明 `hasattr` 的用法和效果：

背景

某函数输入的参数 `rectLocation` 有多种类型

- `tuple`
  - `(x, y, width, height)`
- `Rect` 的类
  - 有对应的属性 `x` , `y` , `width` , `height`

希望实现

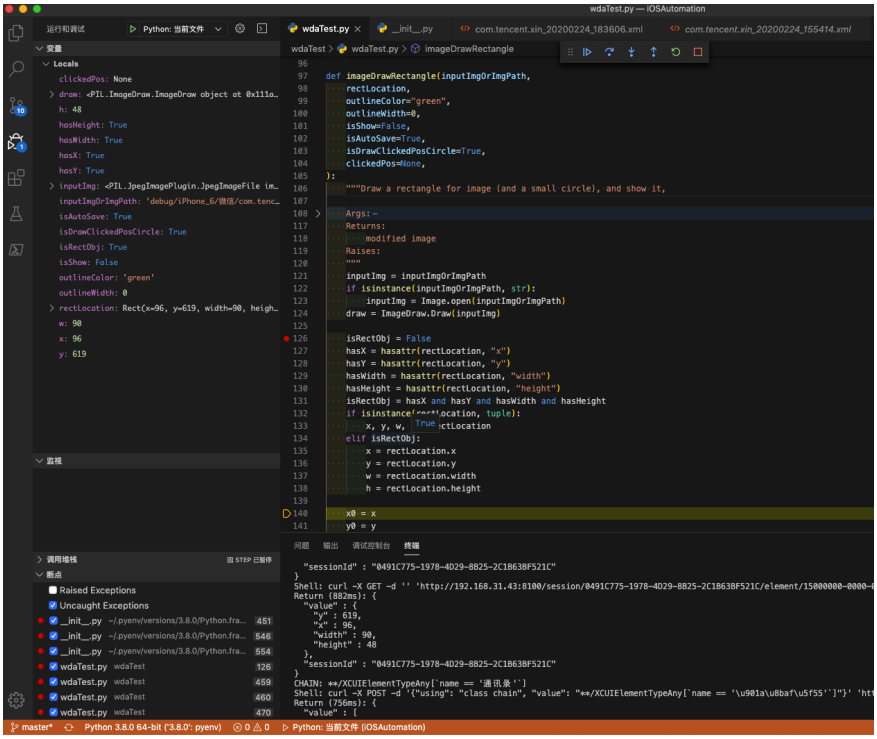
动态判断输入的参数是否有x,y,width,height的属性，从而得到变量的类型

如果有这些属性，再获取属性

否则就是tuple，直接获取值

实现代码

```
def imageDrawRectangle(inputImgOrImgPath,
    rectLocation,
'''
):
'''
    Args:
'''
        rectLocation (tuple/Rect): the rectangle location, (x, y, width, heigh
'''
    """
'''
    isRectObj = False
    hasX = hasattr(rectLocation, "x")
    hasY = hasattr(rectLocation, "y")
    hasWidth = hasattr(rectLocation, "width")
    hasHeight = hasattr(rectLocation, "height")
    isRectObj = hasX and hasY and hasWidth and hasHeight
    if isinstance(rectLocation, tuple):
        x, y, w, h = rectLocation
    elif isRectObj:
        x = rectLocation.x
        y = rectLocation.y
        w = rectLocation.width
        h = rectLocation.height
'''
```



完整代码详见：

crifanLibPython/crifanMultimedia.py at master · crifan/crifanLibPython

附录

- 官网文档：
  - Built-in Functions — Python 3.8.2rc2 documentation

    > hasattr(object, name)
    >
    > The arguments are an object and a string. The result is True if the string is the name of one of the object's attributes, False if not. (This is implemented by calling getattr(object, name) and seeing whether it raises an AttributeError or not.)

  - 另外几个相关函数

> getattr(object, name[, default])
>
> Return the value of the named attribute of object. name must be a string. If the string is the name of one of the object’s attributes, the result is the value of that attribute. For example, getattr(x,'foobar') is equivalent to x.foobar. If the named attribute does not exist, default is returned if provided, otherwise AttributeError is raised.
>
> delattr(object, name)
>
> This is a relative of setattr(). The arguments are an object and a string. The string must be the name of one of the object’s attributes. The function deletes the named attribute, provided the object allows it. For example, delattr(x, 'foobar') is equivalent to del x.foobar.
>
> setattr(object, name, value)
>
> This is the counterpart of getattr(). The arguments are an object, a string and an arbitrary value. The string may name an existing attribute or a new attribute. The function assigns the value to the attribute, provided the object allows it. For example, setattr(x, 'foobar', 123) is equivalent tox.foobar = 123.

Python专题

此处介绍Python相关的、某些具体方面的内容。

字符串

`Python 3` 中的字符串是 `str` 类型。

此处整理和 `str` 相关的内容。

多行字符串

一般写法：

```
normal_multiple_line_str = """line 1
line 2
line 3
"""
```

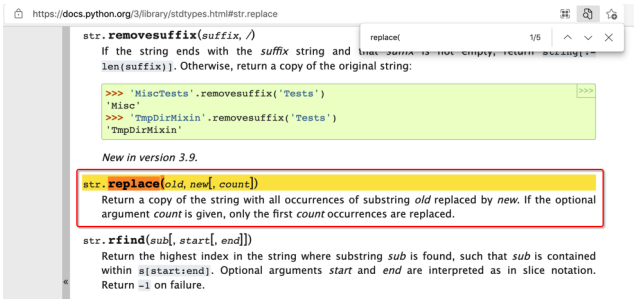
另外一种写法：

```
my_very_big_string = (
    "For a long time I used to go to bed early. Sometimes, "
    "when I had put out my candle, my eyes would close so quickly "
    "that I had not even time to say “I’m going to sleep.”"
)
```

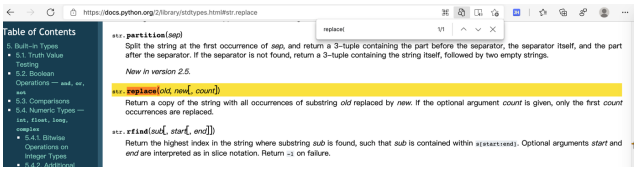
注：多行字符串往往也被用于注释

`str.replace`

`Python` 中用来实现字符串替换的是 `str.replace`

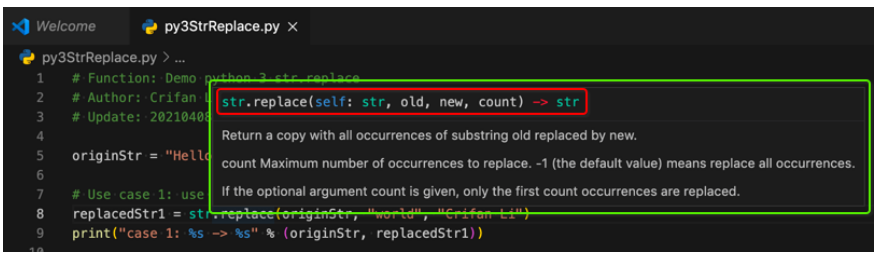
- 语法
  - 官网文档
    - `Python 3`
      - https://docs.python.org/3/library/stdtypes.html#str.replace
      - 

      > str.replace(old, new[, count])
      >
      > Return a copy of the string with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced

    - 对比：`Python 2` 的官网文档，解释也基本类似：
      - https://docs.python.org/2/library/stdtypes.html#str.replace
      - 

      > str.replace(old, new[, count])
      >
      > Return a copy of the string with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced

对应着，VSCode中的语法提示是：



对应着，常见的，具体的用法，有2种：

- 使用方式1：直接用str的replace，即：`str.replace(strVariable, old, new[, count])`

```
replacedStr1 = str.replace(originStr, "from", "to")
```

- 使用方式2：用字符串变量，即：`strVariable.replace(old, new[, count])`

```
replacedStr2 = originStr.replace("from", "to")
```

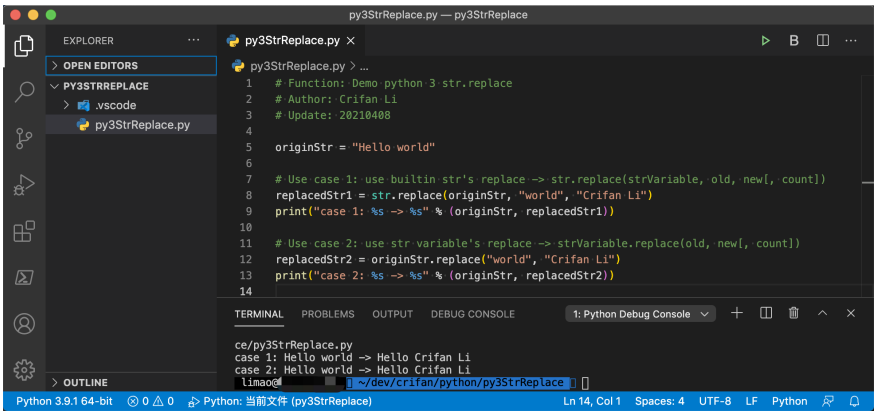
完整demo代码：

```
# Function: Demo python 3 str.replace
# Author: Crifan Li
# Update: 20210408

originStr = "Hello world"

# Use case 1: use builtin str's replace -> str.replace(strVariable, old, new[,
replacedStr1 = str.replace(originStr, "world", "Crifan Li")
print("case 1: %s -> %s" % (originStr, replacedStr1))

# Use case 2: use str variable's replace -> strVariable.replace(old, new[, cou
replacedStr2 = originStr.replace("world", "Crifan Li")
print("case 2: %s -> %s" % (originStr, replacedStr2))
```

输出：

```
case 1: Hello world -> Hello Crifan Li
case 2: Hello world -> Hello Crifan Li
```

效果：



str.replace中的old和new，不支持正则的语法

所以之前的代码：

`sortedUrlList.sort(key=str.replace("^https?://", "", ))`

虽然能运行，但是逻辑不对。只是把 `"^https?://"` 当成了普通字符串而已 -》没有实现希望的正则替换的效果

文件路径

文件（夹）路径，也属于字符串中的一种。

对于早期的 `Python 2` ，对于字符串的支持，相对不够好：用起来要比较注意，才不会犯路径导致的各种错误。

此处整理出来，供借鉴参考。

路径写法不对导致库无法加载文件而报错

比如之前的：

【已解决】Python中出错：pywintypes.com_error,Exception occurred,Microsoft Excel,could not be found – 在路上

遇到的：

如果文件路径写法不对：

即使用看起来正确的绝对路径：

```
xlsPath = "D:\tmp\tmp_dev_root\python\excel_chart\chart_demo.xls"
absPath = os.path.abspath(xlsPath)
wb = xl.Workbooks.open(absPath)
```

也会导致错误： `pywintypes.com_error`

而实际上出错的原因是：此处的字符串的反斜杠，没有转义

即，当字符串中包含反斜杠字符串本身 `\` 时，需要用反斜杠去转义：

```
xlsPath = "D:\\tmp\\tmp_dev_root\\python\\excel_chart\\chart_demo.xls"
```

才可以。

如果不想要把每个反斜杠都写2遍，则可以加上前缀 `r`

```
rawPath = r"D:\tmp\tmp_dev_root\python\excel_chart\chart_demo.xls"
wb = xl.Workbooks.open(rawPath)
```

正则re

Python中的正则表达式的库是内置的 `re`

详见独立教程：

- Python中正则表达式：re模块详解
- 相关应用
  - 正则表达式应用举例

`iterator` 迭代器

迭代是Python最强大的功能之一，是访问集合元素的一种方式。

迭代器 `iterator` 是一个可以记住遍历的位置的对象。

常见问题

iterator对象被访问过一次后，就无效了，值就变成空了

比如对于下面的代码

```
matchIterator = re.finditer(singleScriptPattern, allLine, flags=re.I | re.
print("matchIterator=%s" % matchIterator)
if matchIterator:
  for scriptNum, eachScriptMatch in enumerate(matchIterator):
    print("[%d] eachScriptMatch=%s" % (scriptNum, eachScriptMatch))
    singleScript = eachScriptMatch.group("singleScript")
    print("singleScript=%s" % singleScript)
```

其中的：

`if matchIterator:`

本意是：

判断 `re.finditer` 不为空，然后后续用 `for` 循环获取每个值

结果却变成了：

用 `if` 去判断 `matchIterator` 后，就是对 `matchIterator` 的一次操作，然后 `iterator` 类型的 `matchIterator` 的值就变成空了

就无效了。

->后续代码当然也就不起效果了。

dict字典

下面整理 `dict` 相关内容

删除dict中的某个key值

比如

```
someDict = {
    "someKey": "someValue",
    "otherKey": "otherValue"
}
```

想要删除 `someKey`，则可以写成：

- 普通写法

  ```
  del someDict["someKey"]
  ```

- 更 `Pythonic` 的写法

  ```
  someDict.pop("someKey")
  ```

要确保被删除的key是存在的

否则删除时会出现 `KeyError`

可以先判断存在 `key` 是否存在，再去删除，比如：

```
keyToDelete = "someKey"
if keyToDelete in someDict:
    someDict.pop(keyToDelete)
```

dict的递归的合并更新

背景：希望合并2个dict（json）值

代码：

```
def recursiveMergeDict(aDict, bDict):
    """
    Recursively merge dict a to b, return merged dict b
    Note: Sub dict's won't be overwritten but also updated/merged
    """
    aDictItems = None
    if (sys.version_info[0] == 2): # is python 2
      aDictItems = aDict.iteritems()
    else: # is python 3
      aDictItems = aDict.items()

    for aKey, aValue in aDictItems:
      print("------ [%s]=%s" % (aKey, aValue))
      if aKey not in bDict:
        bDict[aKey] = aValue
      else:
        bValue = bDict[aKey]
        print("aValue=%s" % aValue)
        print("bValue=%s" % bValue)
        if isinstance(aValue, dict):
          recursiveMergeDict(aValue, bValue)
        elif isinstance(aValue, list):
          aValueListLen = len(aValue)
          bValueListLen = len(bValue)
          bValueListMaxIdx = bValueListLen - 1
          for aListIdx in range(aValueListLen):
            print("---[%d]" % aListIdx)
            aListItem = aValue[aListIdx]
            print("aListItem=%s" % aListItem)
            if aListIdx &lt;= bValueListMaxIdx:
              bListItem = bValue[aListIdx]
              print("bListItem=%s" % bListItem)
              recursiveMergeDict(aListItem, bListItem)
            else:
              # recursiveMergeDict(aListItem, aListItem)
              print("bDict=%s" % bDict)
              print("aKey=%s" % aKey)
              print("aListItem=%s" % aListItem)
              bDict[aKey].append(aListItem)

    return bDict

bookJson = recursiveMergeDict(templateJson, copy.deepcopy(currentJson))

pprint("-a"*40)
pprint(templateJson)
pprint("-b"*40)
pprint(currentJson)
pprint("-c"*40)
pprint(bookJson)
```

输出：

```
'-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-a-
{u'pluginsConfig': {u'github-buttons': {u'buttons': [{u'count': True,
                                                       u'repo': u'gitbook_name'
                                                       u'size': u'small',
                                                       u'type': u'star',
                                                       u'user': u'crifan'},
                                                      {u'count': False,
                                                       u'size': u'small',
                                                       u'type': u'follow',
                                                       u'user': u'crifan',
                                                       u'width': u'120'},
                                                      {u'key1': u'string1',
                                                       u'key2': 999,
                                                       u'key3': True}]}}}
'-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-b-
{u'pluginsConfig': {u'github-buttons': {u'buttons': [{u'repo': u'youdao_note_s
'-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-c-
{u'pluginsConfig': {u'github-buttons': {u'buttons': [{u'count': True,
                                                       u'repo': u'youdao_note_s
                                                       u'size': u'small',
                                                       u'type': u'star',
                                                       u'user': u'crifan'},
                                                      {u'count': False,
                                                       u'size': u'small',
                                                       u'type': u'follow',
                                                       u'user': u'crifan',
                                                       u'width': u'120'},
                                                      {u'key1': u'string1',
                                                       u'key2': 999,
                                                       u'key3': True}]}}}
```

实现了希望看到的效果：

- 合并a和b的dict
  - 如果包含子dict，则递归去合并
  - 如果包含list，则针对每个item去合并
    - 如果a的list比b多，则保留a多出的部分

注：最新代码已发布到：

https://github.com/crifan/crifanLibPython/blob/master/python3/crifanLib/crifanDict.py

中的 `recursiveMergeDict`

把dict的json输出到文件且带缩进和不要unicode的 `\uxxxx`

背景： `currentJson.json` 中包含

```
{
  "pluginsConfig": {
    "toolbar-button": {
      "url": "http://book.crifan.com/books/youdao_note_summary/pdf/youdao_note
    },
    "sitemap-general": {
      "prefix": "https://book.crifan.com/gitbook/youdao_note_summary/website/"
    },
    "github-buttons": {
      "buttons": [
        {
          "repo": "youdao_note_summary"
        }
      ]
    }
  },
  "description": "\u603b\u7ed3\u4e4b\u524d\u4f7f\u7528\u8fc7\u6709\u9053\u4e91
  "title": "\u6709\u9053\u4e91\u7b14\u8bb0\u548c\u4e91\u534f\u4f5c\u4f7f\u7528
}
```

`\uxxxx` 的特殊字符

希望导入后能正常解析成中文

最终代码：

```
import codecs

with codecs.open("currentJson.json", 'w', encoding="utf-8") as tmpFp:
    json.dump(currentJson, tmpFp, indent=2, ensure_ascii=False)
```

logging日志

设置File文件的输出格式

用：

```
fileFormatter = logging.Formatter(
    fmt=fileLogFormat,
    datefmt=fileLogDateFormat
)
fileHandler.setFormatter(fileFormatter)
```

即可。

最新代码详见：

https://github.com/crifan/crifanLibPython/blob/master/python3/crifanLib/crifanLogging.py

copy拷贝

copy 和 deepcopy

python中的值，比如一个 `dict` ，如果用了 `copy`

```
copiedDict = originDict.copy()
```

则根据官网文档：

8.17. copy — Shallow and deep copy operations — Python 2.7.15 documentation

则 `copiedDict` 叫做 `shallow copy` = 影子拷贝

-> 类似于 `C` 语言的指针

-> 修改了 `copiedDict` ，原先的 `originDict` 也同时被修改

而想要脱离关系，则需要用到**深度拷贝**

```
import copy
deepCopiedDict = copy.deepcopy(originDict)
```

则 `deepCopiedDict` 和 `originDict` 就没有关系了

-> 修改 `deepCopiedDict` ，**不会影响**到 `originDict` 。

import导入

相对路径导入

此处目录结构是：

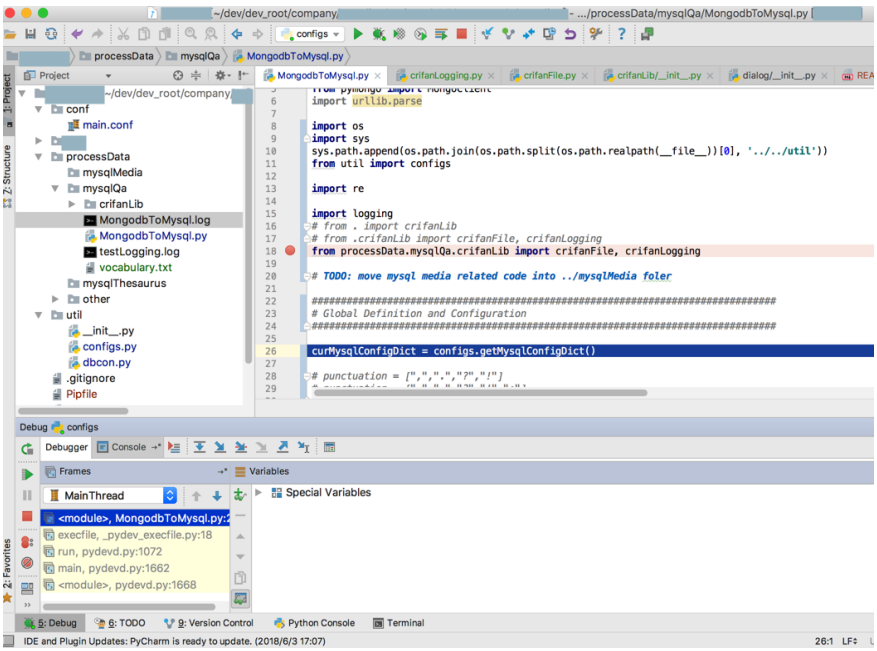
```
devRoot
    processData
        __init__.py
        mysqlQa
            __init__.py
            crifanLib
                __init__.py
                crifanFile
                crifanLogging
```

> 后记：无意间发现：
>
> 去掉 `processData` 下面的 `__init__.py`
>
> 去掉 `mysqlQa` 下面的 `__init__.py`
>
> 也是不影响后续的导入的

写法：

```
from processData.mysqlQa.crifanLib import crifanFile, crifanLogging
```

即可导入：



递归导入 = 循环导入 的问题

此处文件目录结构是：

```
(xx-VJ297lRu) [root@xxx-general-01 crifanLib]# tree -CF
.
├── crifanBeautifulsoup.py
├── crifanCookie.py
├── crifanDatetime.py
├── crifanEmail.py
├── crifanFile.py
├── crifanGeography.py
├── crifanHtml.py
├── crifanHttp.py
├── crifanList.py
├── crifanLogging.py
├── crifanMath.py
├── crifanMysql.py
├── crifanOpenpyxl.py
├── crifanString.py
├── crifanSystem.py
├── crifanTemplate.py
├── crifanUrl.py
└── __init__.py

0 directories, 18 files
(xx-VJ297lRu) [root@xx-general-01 crifanLib]# pwd
/root/xx/nlp/xx/util/crifanLib
(xx-VJ297lRu) [root@xx-general-01 crifanLib]#
```

其中，存在循环=递归导入的问题：

- `crifanFile` 导入了 `crifanList`
- `crifanList` 导入了 `crifanString`
- `crifanString` 导入了 `crifanHttp`
- `crifanHttp` 导入了 `crifanFile`

此处解决递归导入的办法，有几种：

方法1：用的时候再导入

具体操作：把文件顶部的导入，移动到具体文件内部用到别人库的函数的地方，再导入

缺点：麻烦，不利于直观看到引用了哪些库

方法2：改为 `from someOtherLib import someFunction`

缺点：如果用到多个函数，要写多个

据说Python官网中竟然不推荐from xxx import yyy

之前好像在哪里看到Python官网中好像说是：

- 建议用：import X
- 不是很建议用：

```
from module import *
from module import a,b,c
```

和我之前的理解，有偏差，以为：

- 为了性能更好，不要一次性全部导入 `import xxx`
- 而应该用： `from xxx import yyy`

呢

方法3：写成带模块名的 绝对路径 的导入

把：

```
from . import crifanList
```

改为：

```
import crifanLib.crifanList
```

即可。此处最后采用此方式。

类和对象

继承

举例：用Python实现类的**继承**

```
from pyspider.result import ResultWorker

class AutohomeResultWorker(ResultWorker):

    def __init__(self, resultdb, inqueue):
        """init mysql db"""
        print("AutohomeResultWorker init")
        print("resultdb=%s, inqueue=%s" % (resultdb, inqueue))
        # init parent
        ResultWorker.__init__(self, resultdb, inqueue)

        # init current
        # print("self.mysqldb=%s" % (self.mysqldb))
        # if self.mysqldb is None:
        self.mysqldb = MysqlDb()
        print("self.mysqldb=%s" % self.mysqldb)
```

site-packages

`site-packages` 是个 `文件夹` = `目录`，用来存放 `Python` 的第三方库。

换句话说，Python第三方库安装后，往往都可以在对应的 `site-packages` 中找到。

找到当前Python的site-packages的位置

背景：安装了个Python库 `Django`，想要知道当前对应安装到哪里了，即找到对应的 `site-packages` 的位置

方法：

```
python3 -m site
```

或：

```
pip3 show Django
```

即可找到处 `Django` 所在的 `Python3` 的 `site-packages` 的位置是：

`/usr/lib/python3.4/site-packages`

自带IDE：IDLE

`Python` 安装后，往往自带一个 `IDE`，叫做 `IDLE`

IDEL对新手不友好，所以不推荐

不过由于其对于新手不够友好，甚至有时候会误导新手。所以不建议作为第一个使用的 `Python` 的IDE去使用

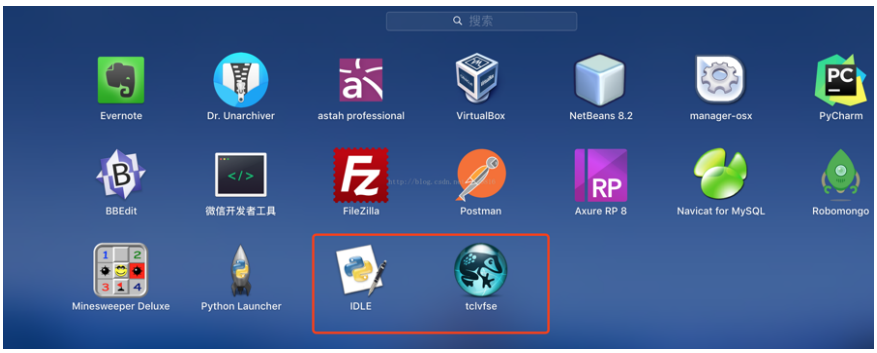
还是推荐新手用 `VSCode` 或 `PyCharm`，详见：Python的IDE · 让你人生不苦短的编程语言：Python

虽然不推荐，但是也简单介绍一下，供需要的了解。

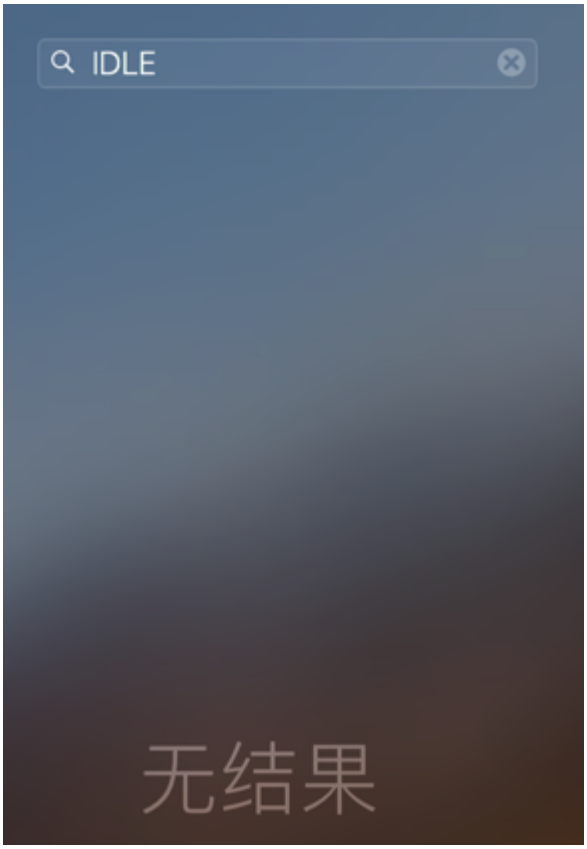
Mac中的 `IDLE`

Mac中安装Python后，正常也是有IDLE的。

可以在 `启动台` = `Launcher` 中看到：



但是我这里却找不到：



不过可以去命令行终端中找到：

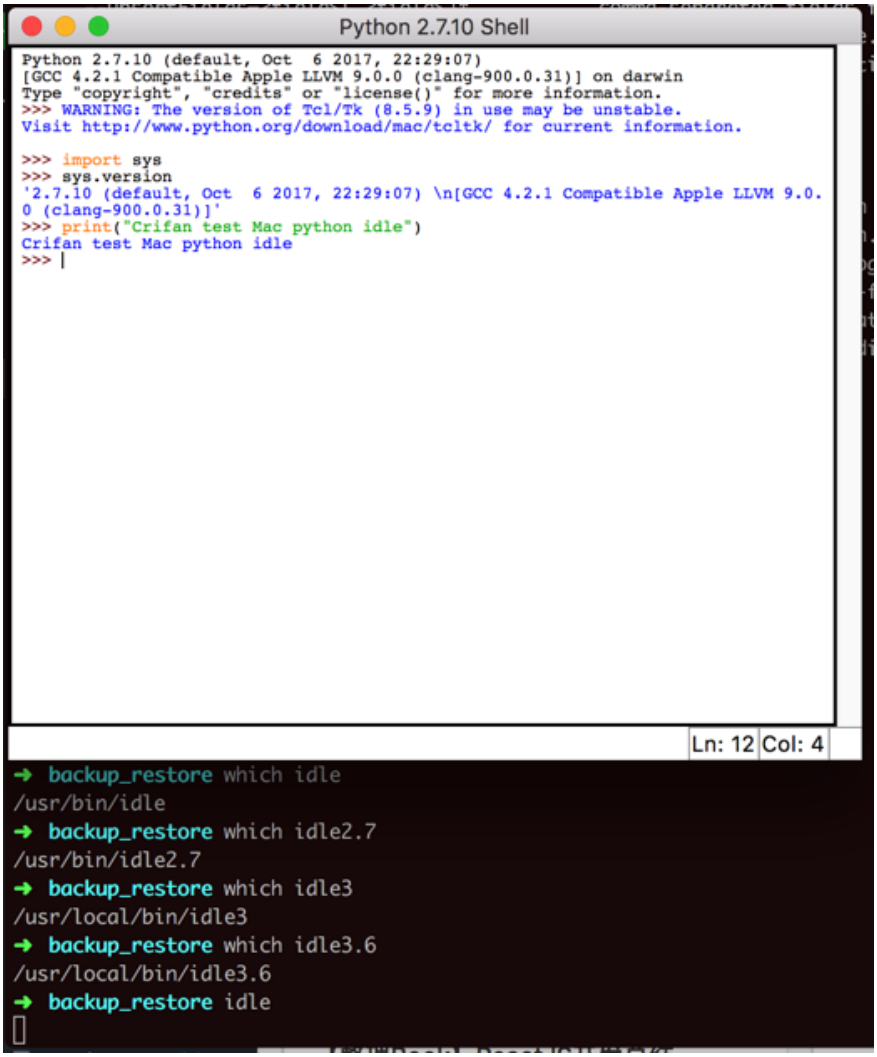
此处有多个Python版本，对应多个版本的 `idle`

```
➜  backup_restore which idle
/usr/bin/idle
➜  backup_restore which idle2.7
/usr/bin/idle2.7
➜  backup_restore which idle3
/usr/local/bin/idle3
➜  backup_restore which idle3.6
/usr/local/bin/idle3.6
```

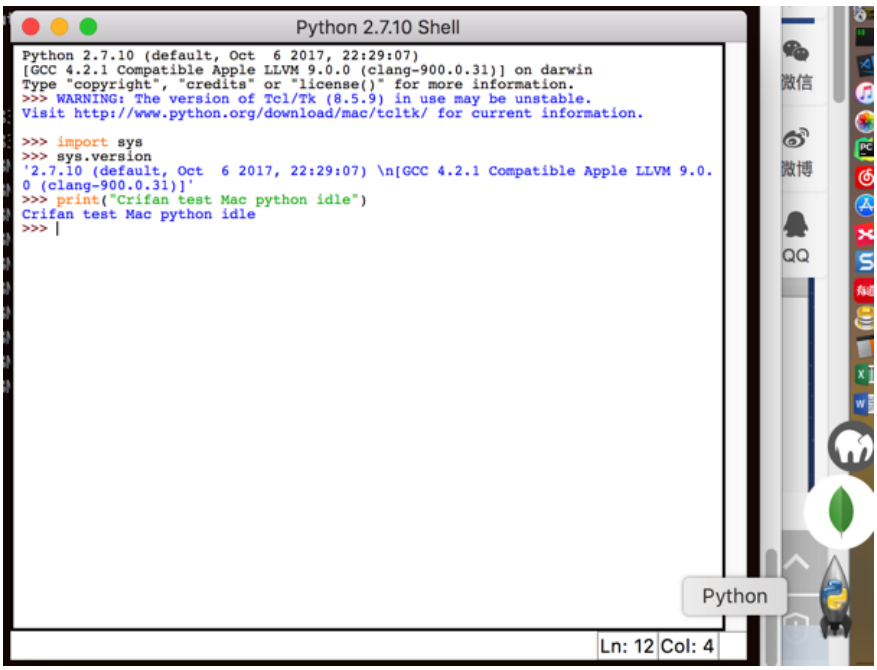
输入：

```
idle
```

以启动Mac中的 `IDLE` 看看是啥效果：



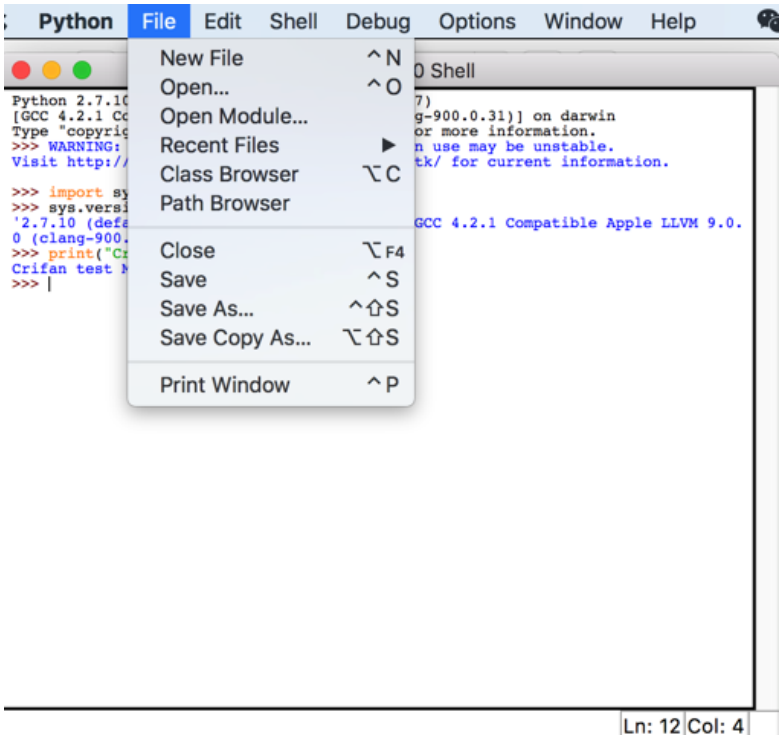
Docker中可以看到图标是`Python`：

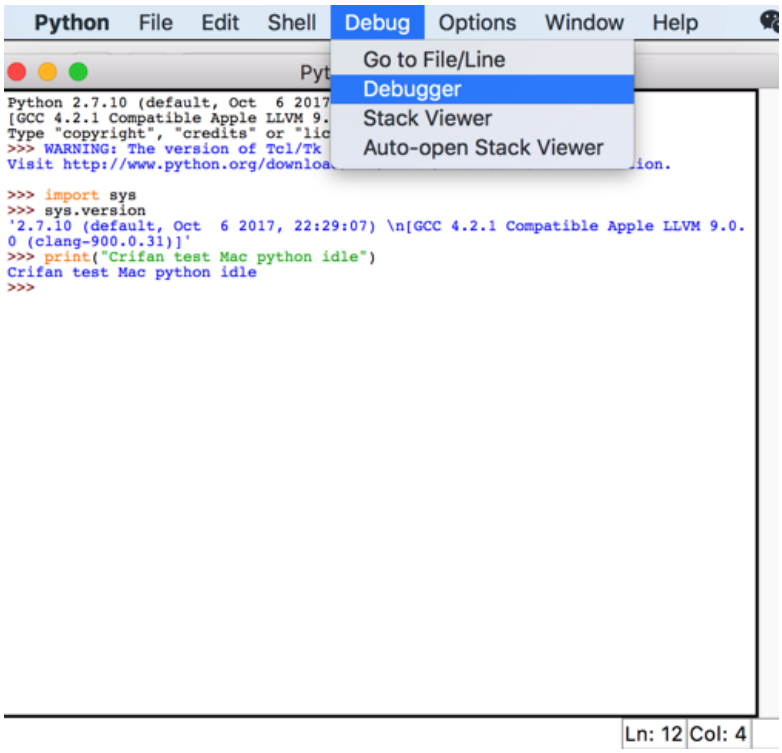


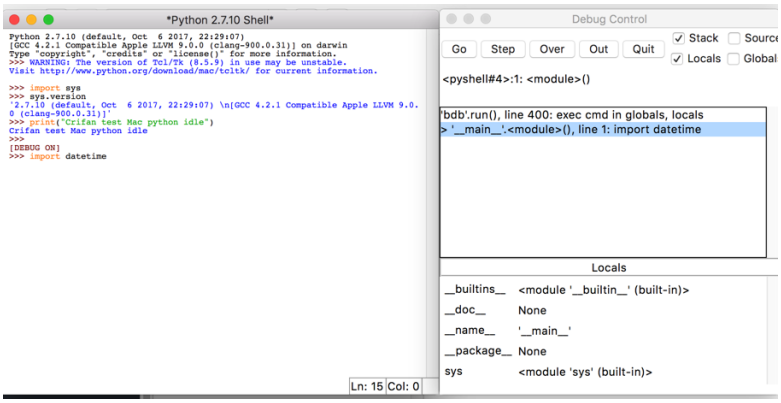
基本上可以理解为：

`python` 的 `shell` 的彩色版本

当然作为IDE，也是有基本的功能的：

- File
  - 

- Debug
  - 



总体来说，还是很难用的。

另外， `Control+C` 去（强制）退出时，还报错了：

```
➜  backup_restore idle
*** Internal Error: rpc.py:SocketIO.localcall()
Object: gui_adapter
Method: <bound method GUIAdapter.interaction of <idlelib.RemoteDebugger.GUIAda
Args: ('<pyshell#4>:1: <module>()', 4397130240, None)

Traceback (most recent call last):
  File "/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7
    ret = method(*args, **kwargs)
  File "/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7
    self.gui.interaction(message, frame, modified_info)
  File "/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7
    b.configure(state="disabled")
  File "/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7
    return self._configure('configure', cnf, kw)
  File "/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7
    self.tk.call(_flatten((self._w, cmd)) + self._options(cnf))
TclError: invalid command name ".4576786048.4576788064.4576787920"
[1]    52979 terminated  idle
```

忽略之，当然也懒得继续研究了。

多线程

Python也支持多线程。

- 进程和线程
  - （多）进程
    - 系统分配资源的最小单位
    - python库：multiprocessing
      - from multiprocessing import Process
    - 缺点
      - 进程间共享资源不方便
        - 比如变量共享
      - 资源开销比较大
        - 需要先创建，用完再销毁
      - 进程间切换比较耗时
  - （多）线程
    - python库：threading
      - from threading import Thread
    - 特点
      - 进程间共享资源方便 = 简单
      - 开销小
        - 单个进程中可以创建多个线程
  - 对比
    - 

      - CPU密集型
      - I/O密集型
        - 结论：用协程
        - 

        -
        - 协程：

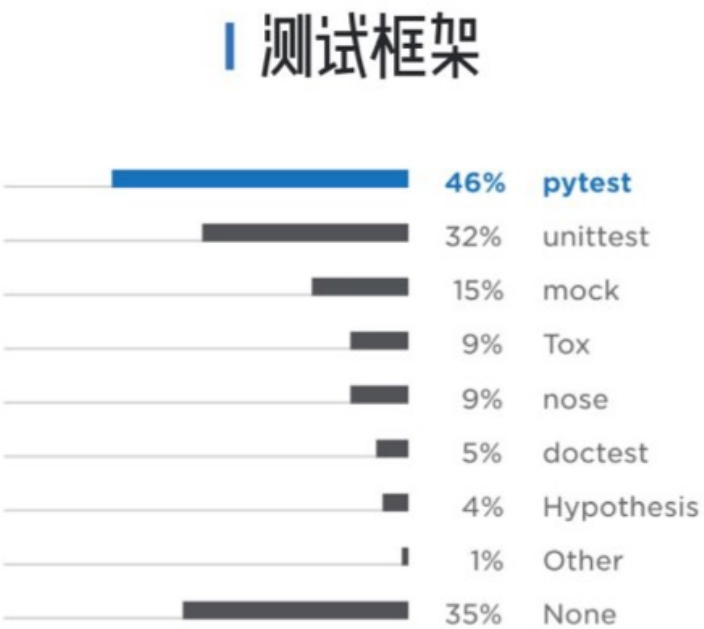
- 用户态轻量级的线程
- 有自己的寄存器和上下文切换
- 比线程还要更轻量

期间涉及到：

- `全局解释器锁` = `GIL` = `Global Interpreter Lock`

测试

- Python测试相关框架有：
  - `pytest`
  - `unittest`
- 详见



领先的单元测试框架是 pytest，然后是 unittest。
其他单元测试框架远没那么受欢迎。
令人惊讶的是，35%的Python用户不使用任何测试框架，并且可能没有测试他们的代码。

历史版本

Python进过多年发展，现在已有很多历史版本。

简单总结相关内容如下。

Python 3.7

`Python 3.7` 比之前有很大进步，其中相对较大的进步有：

- 支持dict插入时保存顺序
  - > dict 对象的插入顺序保存特性已被声明为 Python 官方的语言规范中
- 默认编码不再是ASCII，避免是ASCII，而导致的很多文件编解码时候的错误

  > CPython 实现改进：
  >
  > - 避免使用 ASCII 作为默认文本编码
  > - PEP 540：强制 UTF-8 模式

Python各技术领域

最后更新： 2021-04-10 11:29:49

版本和环境

- Python
  - 多版本共存 -> `pyenv`
    - 多版本共存
      - 多个主版本：`Python 2.x` 和 `Python3.x`
      - 同一主版本内多个小版本：`Python 3` 的 `Python 3.6` 、 `Python 3.7` 等
  - 隔离不同项目的Python环境 = **虚拟环境** -> `virtualenv` 、 `pipenv` 等
    - 专门处理科学计算相关 -> `conda`

`Python 2` 和 `Python 3` 的兼容

如果想要写代码，同时 `支持` = `兼容` Python 2 `和` Python 3`，则可以考虑用专门的库：

- `six`
  - Github
    - Python 2 and 3 compatibility library
      - https://github.com/benjaminp/six
  - 文档
    - Six: Python 2 and 3 Compatibility Library — six 1.14.0 documentation
      - https://six.readthedocs.io/

不过也需要了解到：

> 虽然利用 `__future__` 和 `six` ，可以写出同时兼容 `Python 2/3` 的程序。
>
> 但我觉得这样看上去未免会使程序变得丑陋，而且很多 `Python3` 的新特性还没法用了（例如 `asyncio` , `Type Hinting` , `f-string` 等）
>
> 所以还是希望Python3能够普及

多版本共存

Python的多版本共存，一般用 `pyenv` 。

即，用 `pyenv` 去管理多个 `Python` 版本。

虚拟环境

实际的Python环境中，往往会用到多个不同的Python版本，以及每个项目中往往用到不同的库，且版本不同。

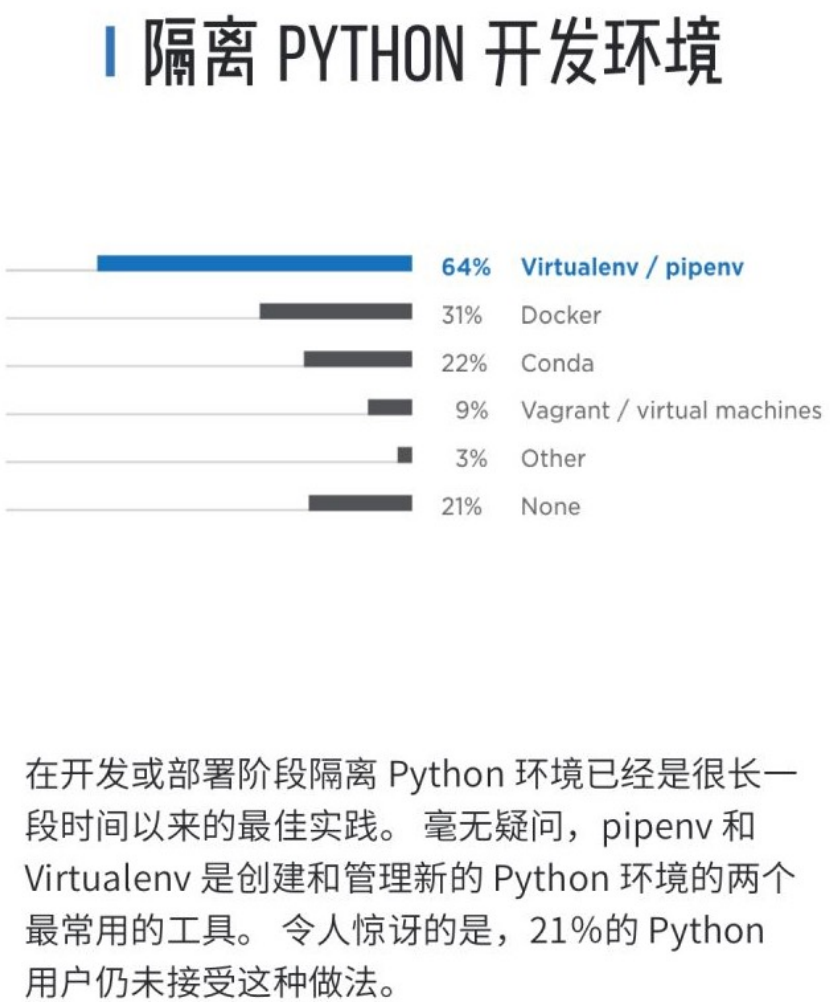
如果都安装到一个环境中，就会导致混乱和冲突，无法正常开发。

对此，可以用Python的 `虚拟环境` 去解决：隔离不同项目中的Python版本和依赖的库。

Python的虚拟环境工具，目前常见的有：

- `Virtualenv / Pipenv`
- `Docker`
- `Conda`
- 等

如图：



详见独立教程：

隔离Python项目环境：虚拟环境

表格处理

可以用Python处理`csv`、`excel`等表格相关文件。

详见独立教程：

Python表格处理：CSV和Excel

Web框架

用Python开发Web网络应用，尤其是后端服务，也有很多Web框架库供参考。

所以基本可以理解为：

- Python的Web框架 = Python的后端框架

下面整理一下相关框架：

- Python后端框架
  - 主流的
    - `Flask`
      - 详见：
        - 轻量但强大的Python框架：Flask
    - `Django`
      - 功能很强，但有些重量级
  - 其他不够流行的
    - `web.py`
      - 之前用过的宝塔面板(好像）就是基于web.py去搭建的
        - 升级宝塔Linux面板中的log输出中看到过 `web.py`

```
Requirement already up-to-date: pip in /usr/lib/python
Requirement already satisfied: psutil in /usr/lib64/py
Requirement already satisfied: chardet in /usr/lib/pyt
Requirement already satisfied: web.py in /usr/lib/pyth
Requirement already satisfied: virtualenv in /usr/lib/
```

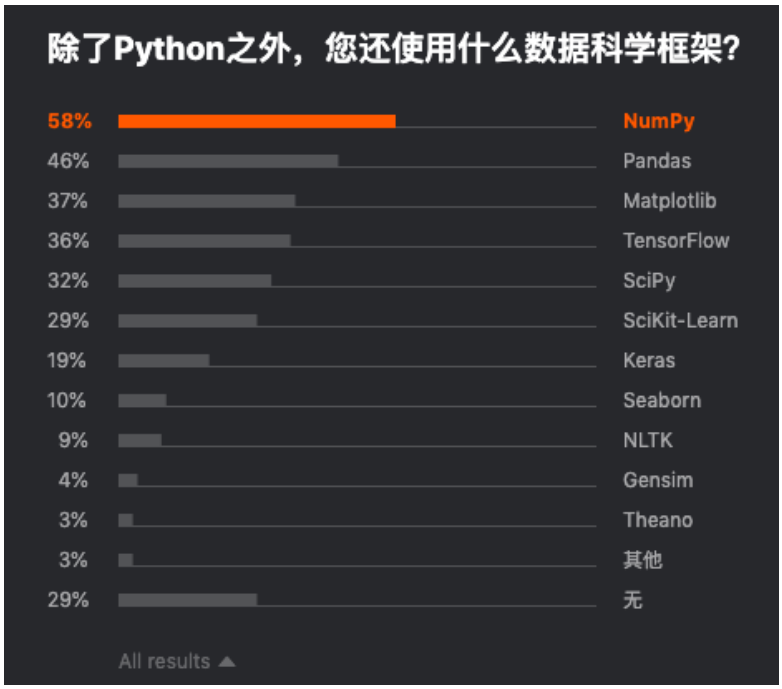
    - `Tornado`
    - `Bottle`

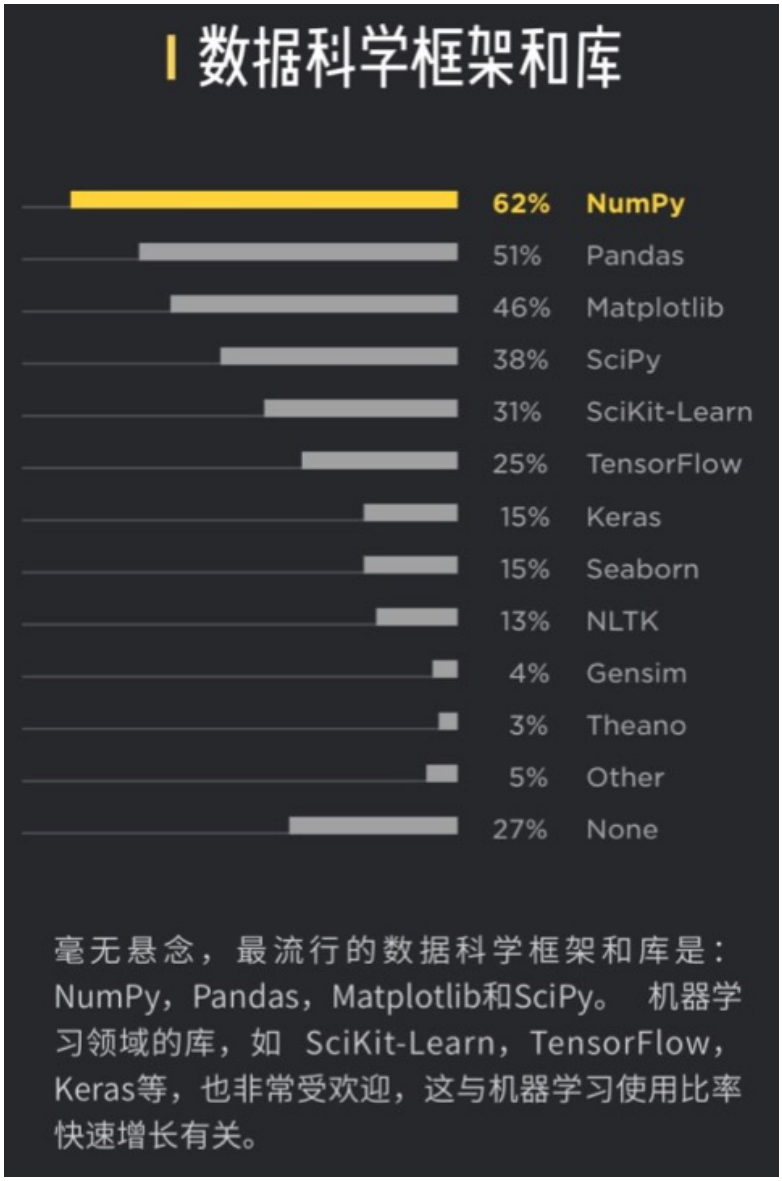
详见：


PYTHON 框架、库和技术
WEB 框架
47% Flask
45% Django
6% Tornado
4% Pyramid
4% web2py
4% Bottle
3% CherryPy
2% Falcon
1% Hug
1% TurboGears
25% None
令人惊讶的是，与前一年相比，我们调查的受访者中 Flask 的使用量增长了15个百分点，因此，今年 Flask 已成为最受欢迎的Web框架。 45%的受访者（2017年为41%）选择了 Django。

数据科学和人工智能

数据科学框架和库

- 用的比较多的
  - Numpy
  - Pandas
  - Matplotlib
  - Scipy
  - SciKit-Learn
  - TensorFlow
  - Keras
  - Seaborn
  - NLTK
- 详见
  - 



大数据相关的库和框架

- 用的比较多的Python的大数据相关库有
  - Apache Spark
  - Apache Hadoop/MapReduce
  - Apache Kafaka
  - Apache Hive
- 详见

大数据工具更有可能被机器学习工程师使用，这就是为什么76%的受访者没有选择任何工具。 Spark 占据了第一名，随后是 Hadoop 和 Kafka。

。

网络

图像处理

爬虫

详见独立教程：

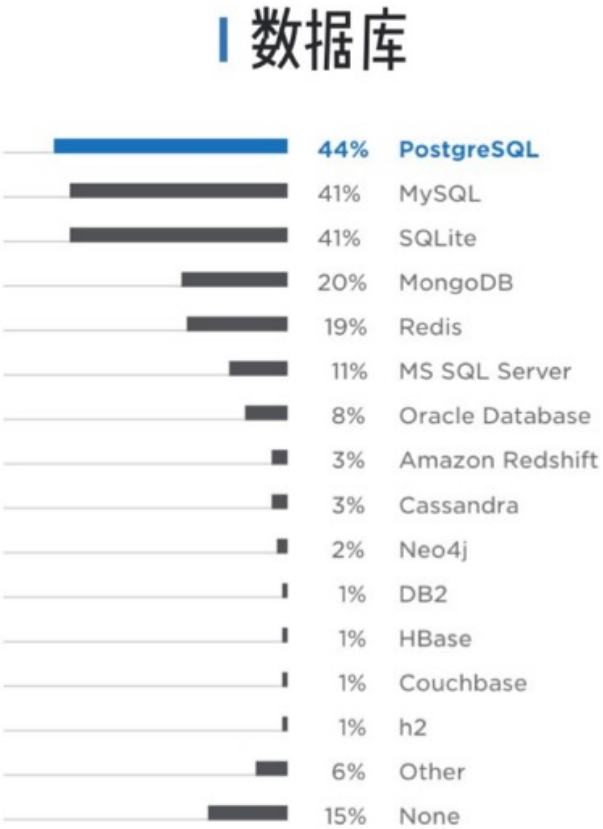
如何用Python写爬虫

相关教程：

爬取你要的数据：爬虫技术

GUI图形界面

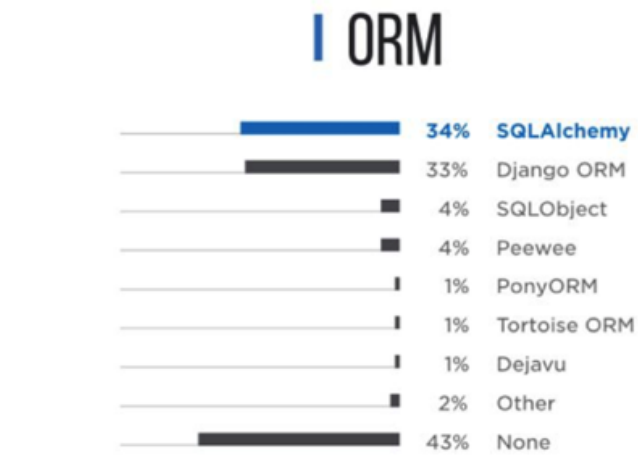
数据库

- 常用的Python能操作的数据库有
  - `MySQL`
  - `MongoDB`
    - 相关教程
      - 主流文档型数据库：MongoDB
  - `Redis`
  - `SQLite`
  - `PostgrelSQL`
  - `Microsoft` 的 `SQL Server`
  - `Oracle`
- 详见
  - 

    大多数人使用免费或开源数据库，如 PostgreSQL，MySQL 或 SQLite。
    MongoDB 和 Redis 等非关系型数据库也非常受欢迎，因为大量的 Python 用户正在进行某种形式的机器学习或数据工程。

用Python操作不同数据库期间，可能会涉及到：`ORM`：

- 常见的 `ORM` 有：

    - SQLAlchemy
    - Django ORM
- 详见
    - 

    两个最流行的 ORM 是 SQLAlchemy 和 Django ORM，它们与两个领先的Web 开发框架：Flask 和 Django 的流行度相匹配。

相关教程：

- 【整理Book】保存数据的仓库：数据库

打包

写好Python代码，实现了功能后，想要给别人用。

除了直接给别人代码外，可以考虑：打包。

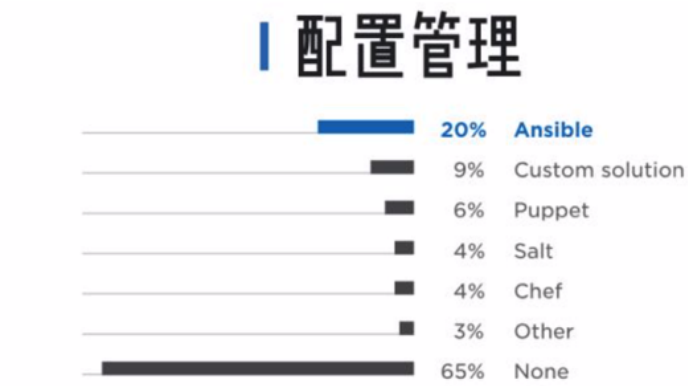
即把Python源码，打包成可执行文件，比如

- `Windows` 中的 `exe`
- `Mac` 中的 `dmg` 、 `app`

等，就可以使用对应工具：

- `PyInstaller`
  - 概述
    - 特点：简单好用，功能强大，历史悠久
    - 用法举例
      - 生成单一的exe文件
        - `pyinstaller.py -F ..\BlogsToWordpress\BlogsToWordpress.py`
      - 添加额外文件 = 添加必要的搜索路径（自动搜索到对应文件）
        - Windows中多个路径用分号 `;` 分隔开
          - `pyinstaller.py -F -p D:\tmp\tmp_dev_root\python\tutorial_summary\make_exe\BlogsToWordpress\libs;D:\tmp\tmp_dev_root\python\tutorial_summary\make_exe\BlogsToWordpress\libs\crifan;D:\tmp\tmp_dev_root\python\tutorial_summary\make_exe\BlogsToWordpress\libs\crifan\blogModules;D:\tmp\tmp_dev_root\python\tutorial_summary\make_exe\BlogsToWordpress\libs\thirdparty;D:\tmp\tmp_dev_root\python\tutorial_summary\make_exe\BlogsToWordpress\libs\thirdparty\chardet; ..\BlogsToWordpress\BlogsToWordpress.py`
      - 添加必要的搜索路径，且带图标
        - `pyinstaller.py -F -p D:\tmp\tmp_dev_root\python\tutorial_summary\make_exe\BlogsToWordpress\libs;D:\tmp\tmp_dev_root\python\tutorial_summary\make_exe\BlogsToWordpress\libs\crifan;D:\tmp\tmp_dev_root\python\tutorial_summary\make_exe\BlogsToWordpress\libs\crifan\blogModules;D:\tmp\tmp_dev_root\python\tutorial_summary\make_exe\BlogsToWordpress\libs\thirdparty;D:\tmp\tmp_dev_root\python\tutorial_summary\make_exe\BlogsToWordpress\libs\thirdparty\chardet; -i ..\BlogsToWordpress\BlogsToWordpress.ico ..\BlogsToWordpress\BlogsToWordpress.py`
  - 详解
    - Python打包利器：PyInstaller
- `Nuitka`
  - 新出的工具

配置管理

- 常用的配置管理库有
  - `Ansible`
  - `Puppet`
- 详见
  - 

    大多数 Python 用户不使用配置管理工具。 在使用者中，第一名显然是 Ansible。

项目部署

此处项目部署相关内容有：

- 上传代码 = 同步代码
  - `Fabric`
    - http://www.fabfile.org/
- 部署Python项目环境
  - `gunicorn`
  - `supervisor`
    - 相关
      - 后台任务
        - `Celery`

gunicorn

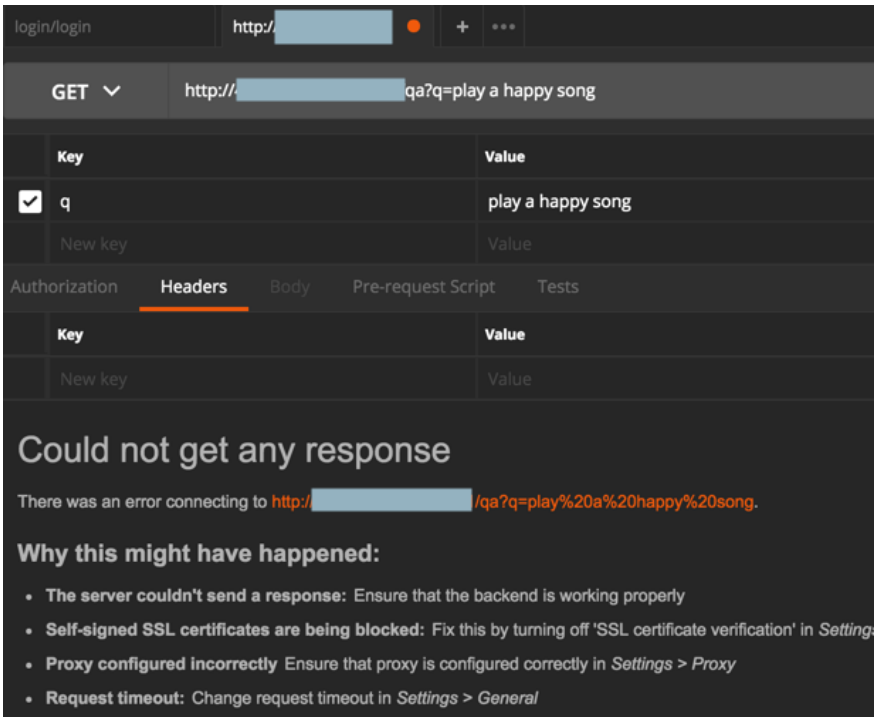
常见问题

`org.apache.http.conn.HttpHostConnectException Connect to failed Operation timed out`

现象：Flask的app，本地用gunicorn去运行是正常的，但是部署到线上后，用客户端去访问api出错：

```
org.apache.http.conn.HttpHostConnectException: Connect to x.x.x.x:port [/x.x.x
```

对应的Postman也无法访问线上端口：



可能原因 + 解决办法 + 操作步骤：

1. 线上环境是 阿里云的服务器。阿里云有个安全组，默认把非常用端口关闭了，导致此处无法访问对应端口。
   - **解决办法**：去阿里云的安全组中开通允许外部访问此端口
     - 

2. `gunicorn` 默认不允许非本机访问端口
   - **解决办法**：修改 `gunicorn` 配置的端口监听，允许外部访问端口
     i. 把 `Flask` 中 `app` 的 `host` 改为 `0.0.0.0`
        - 文件： `app.py`

```
if __name__ == "__main__":
    app.run(
        host="0.0.0.0",
        port=SERVER_PORT,
        debug=DEBUG
    )
```

     ii. 把 `gunicorn` 中的 `host` 改为 `0.0.0.0`
        - 文件： `gunicorn_config.py`

```
#bind = '127.0.0.1:32851' # 绑定ip和端口号
bind = '0.0.0.0:32851' # 绑定ip和端口号
```

supervisor

部署 `celery+redis` 的Python项目

此处把本地 `Mac` 中正常工作的 `celery+redis` 部署到在线 `CentOS` 中的步骤是：

CentOS中安装redis

```
yum -y install redis
```

更新 `supervisord` 的配置

文件： `/etc/supervisord.d/supervisord_server.conf`

```
[program:redis]
directory=/xx/robotDemo
command=/usr/bin/redis-server
autostart=true
autorestart=true
stdout_logfile=/xx/robotDemo/logs/redis-%(program_name)s-stdout.log
stdout_logfile_maxbytes=2MB
stdout_logfile_backups=10
stderr_logfile=/xx/robotDemo/logs/redis-%(program_name)s-stderr.log
stderr_logfile_maxbytes=2MB
stderr_logfile_backups=10

[program:robotDemo_CeleryWorker]
command=/root/.local/share/virtualenvs/robotDemo-dwdcgdaG/bin/celery worker -A
directory=/xx/robotDemo
autostart=true
autorestart=true
stdout_logfile=/xx/robotDemo/logs/celery-worker-%(program_name)s-stdout.log
stdout_logfile_maxbytes=2MB
stdout_logfile_backups=10
stderr_logfile=/xx/robotDemo/logs/celery-worker-%(program_name)s-stderr.log
stderr_logfile_maxbytes=2MB
stderr_logfile_backups=10

[program:robotDemo]
command=/root/.local/share/virtualenvs/robotDemo-dwdcgdaG/bin/gunicorn -c guni
directory=/xx/robotDemo
startsecs=0
stopwaitsecs=0
autostart=true
autorestart=true
killasgroup=true
stopasgroup=true
stdout_logfile=/xx/robotDemo/logs/supervisord-%(program_name)s-stdout.log
stdout_logfile_maxbytes=2MB
stdout_logfile_backups=10
stderr_logfile=/xx/robotDemo/logs/supervisord-%(program_name)s-stderr.log
stderr_logfile_maxbytes=2MB
stderr_logfile_backups=10
```

说明：

- 本地此处之前已经配置好了 `supervisord` 的配置，默认是加载： `/etc/supervisord.conf`
  - 其中最后是有相关逻辑：

    ```
    [include]
    ;files = supervisord.d/*.ini
    files = /etc/supervisord.d/*.conf
    ```

    - 所以保证了 `/etc/supervisord.d/` 中的 `supervisord_server.conf` 能被执行到。
- `supervisord_server.conf` 中的配置顺序是
    - `redis`
    - `celery worker`
        - 其中用到 `redis`
    - `flask` 的 `app`： `robotDemo`
        - 用到 `celery worker + redis`

fabric

- `Fabric` ： 项目代码部署工具
  - 主要功能
    - SSH登录远程服务器，然后执行各种操作命令
      - 偶尔会用到命令操作执行的结果
  - 主要包含3部分
    - Invoke：
      - 主要处理：命令行中的参数导入，任务管理，shell命令执行等事情
      - 对于（此处hello world这种）和远程remote服务器没关系的，往往都是调用invoke实现的
      - 用户常需要导入Invoke，去实现命令解析等工作
    - Paramiko
      - 主要实现了底层和中层的SSH的功能：SSH和SFTP的会话，key的管理等
      - Fabric内部调用Paramiko，用户很少需要import导入Paramiko
    - Fabric
      - 打包了其他各种库和功能
        - 继承了Invoke的Context上下文
        - 封装了Paramiko底层的功能
        - 通过Paramiko的ssh_config去扩展了Invoke的配置系统
        - 实现了高层功能封装，比如端口转发的上下文管理等

使用Fabric之前一定要注意版本

- Fabric有3个版本
  - Fabric1
    - https://pypi.org/project/Fabric/
    - `pip install Fabric`
  - Fabric2
    - https://pypi.org/project/fabric2/
    - `pip install fabric2`
  - Fabric3
    - https://pypi.org/project/Fabric3/
    - `pip install fabric3`

Fabric的3个版本的区别

- Fabric1和Fabric2，在pypi中的页面，就是同一个东西：
  - 都是Fabric的最新版：Fabric 2.x
    - 截至到 `20180817` ，安装出来的版本是： `2.3.1`
  - 而官网之所以弄出来个Fabric2是因为：
  - Fabric2和Fabric1相比，完全重写了，接口和功能都有很大改动
    - 官网也不建议你继续用Fabric1，建议升级到Fabric2
    - 最新版也早就支持Python 3.4+，和之前的Python2.7

- 而Fabric3，是非官网的
  - 是当之前Fabric1还没有支持Python3时，别人去fork出来，加了Python 3的支持的
  - 现在好像基本上不维护了

`Fabric 1` vs `Fabric 2`

- Fabric 1到Fabric 2的改动很多
  - 详见
    - http://www.fabfile.org/upgrading.html#upgrade-specifics
    - http://www.fabfile.org/upgrading.html#api-organization
  - 其中一些是：
    - 导入库的方式
      - `Fabric 1`

        ```
        from fabric.api import *
        ```

      - `Fabric 2`

        ```
        from invoke import task
        from fabric import Connection
        ```

        - 等等，具体情况需要根据之前自己的需求，从invoke，fabric等导入合适的功能模块
    - 任务函数定义
      - `Fabric 1`

        ```
        def doSomething()
        ```

      - `Fabric 2`：要加上 `@task` 修饰符=装饰器

        ```
        @task
        def doSomething()
        ```

如何选择Fabric的版本

- 尽量用最新的Fabric2
  - Mac中用 `brew install fabric`
    - 已经是最新的 `2.3.1` 版本了

      ```
      > fab --version
      Fabric 2.3.1
      Paramiko 2.4.1
      Invoke 1.1.1
      ```

  - 如果 `pip install` 的话，应该是：
    - pip install fabric
    - pip install fabric2
      - 都是最新版本
- 尽量不要用之前旧的版本的 `Fabric1` 了
  - 如果还在用，建议升级到最新的 `Fabric2`
    - Upgrading from 1.x — Fabric documentation

        - http://www.fabfile.org/upgrading.html#why-upgrade
- 不需要操心、忽略掉，所谓的、非官网的，现在已没价值的：`Fabric3`

Fabric文档和资料

- 中文文档
  - 网上找到的（之前官网的）都是`Fabric 1`的
    - 概览 & 教程 — Fabric 文档
      - https://fabric-chs.readthedocs.io/zh_CN/chs/tutorial.html
  - 而最新的 Fabric官网
    - http://www.fabfile.org
      - 中都是最新的`Fabric 2+`的版本的
- 英文文档
  - Welcome to Fabric's documentation! — Fabric documentation
    - http://docs.fabfile.org/en/latest/
  - Getting started — Fabric documentation
    - http://docs.fabfile.org/en/latest/getting-started.html

fab

Fabric安装后，有个命令行工具，叫`fab`

- `fab`
  - 用法
    - Command-line interface — Fabric documentation
      - http://docs.fabfile.org/en/latest/cli.html
  - 参数
    - 首先是支持invoke的inv
      - inv[oke] core usage — Invoke documentation
        - http://docs.pyinvoke.org/en/latest/invoke.html#inv
      - 的所有参数
    - 其次额外加了些自己的参数
    - 默认会加载当前目录下的fabfile.py
  - 常用的类和函数功能
    - run
      - http://docs.fabfile.org/en/2.3/api/connection.html#fabric.connection.Connection.run
        - 是远程的执行，不是local本地
        - 调用的是invoke.runners.Runner.run
        - 注意不是context的run
    - Connection
      - http://docs.fabfile.org/en/2.3/api/connection.html#fabric.connection.Connection
    - Result
      - http://docs.pyinvoke.org/en/latest/api/runners.html#invoke.runners.Result

Fabric的使用

去写代码，一般叫做： `fabfile.py`

比如：

```
from invoke import task

def fabLoadedPath():
    from fabric.main import program
    return program.collection.loaded_from

@task
def upload(context):
    # print("upload: context=%s", context)

    fabFilePath = fabLoadedPath()
    print("fabFilePath=%s" % fabFilePath)
```

然后再去命令行中通过 `fab` 去运行 `fabfile.py` 中的task：

```
fab upload
```

- `upload` 是你的 `task` 的名字
  - =你 `fabfile.py` 中加了 `@task` 的函数名

服务监控

- 服务监控 = 性能监控
  - `OneAPM`

其他Python相关

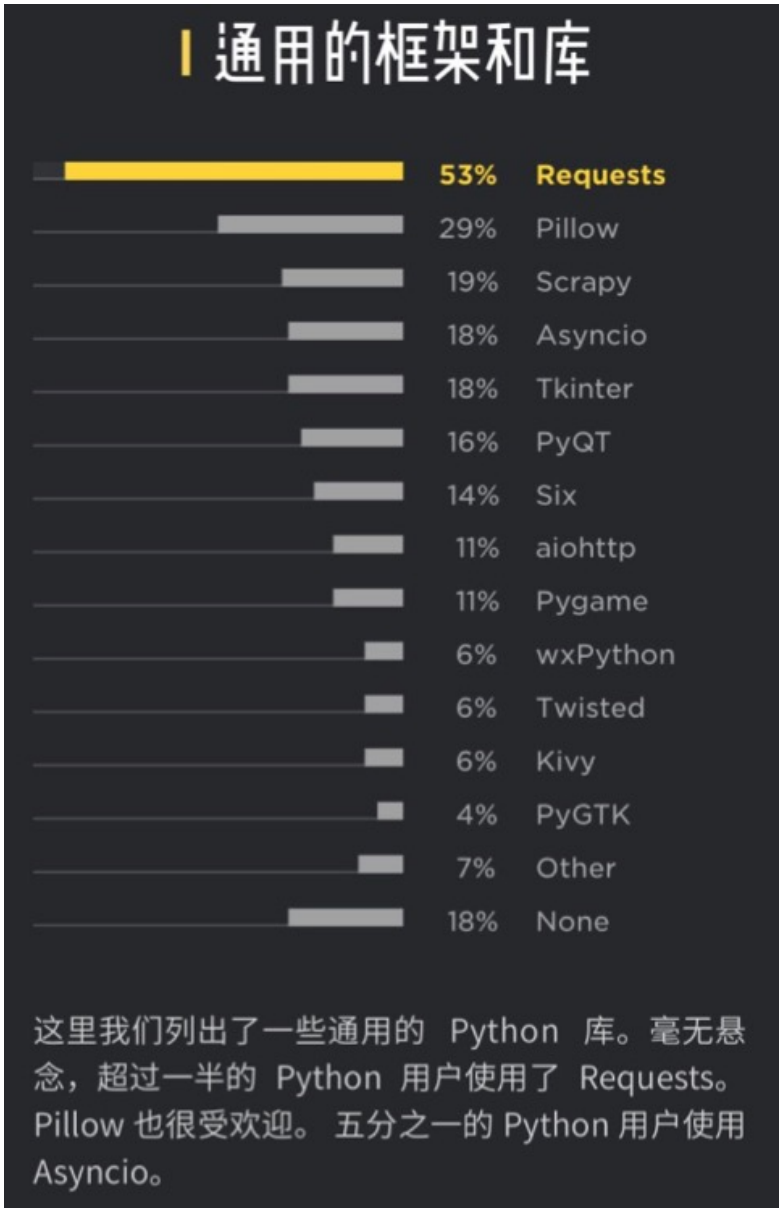
PEP

社区 — The Hitchhiker's Guide to Python

常见的PEP

- PEP 8 -- Style Guide for Python Code
  - PEP 8 -- Style Guide for Python Code | Python.org
- PEP 20 -- The Zen of Python
  - PEP 20 -- The Zen of Python | Python.org

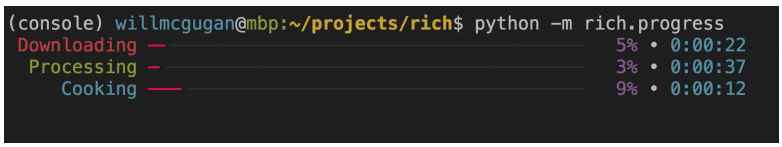
其他常用Python库

- 其他的通用的Python框架和库，还有：
  - Requests
  - Pillow
  - Scrapy
  - Asyncio
  - Thinker
  - PyQT
  - Six
  - aiohttp
  - PyGame
- 详见
  - 

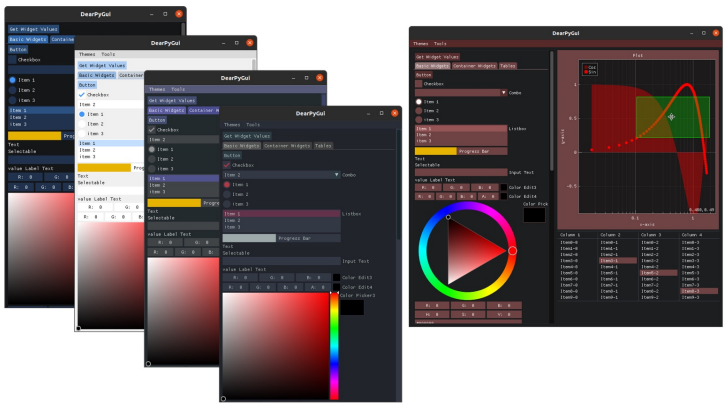
有趣的库

此处整理一下有趣的Python的库

rich

- Github
  - willmcgugan/rich: Rich is a Python library for rich text and beautiful formatting in the terminal.
    - https://github.com/willmcgugan/rich
- 截图
  - 

DearPyGui

- `DearPyGui`
  - 一款新的Python的GUI图形库
  - Github
    - hoffstadt/DearPyGui: DearPyGui: A GPU Accelerated Python GUI Framework
      - https://github.com/hoffstadt/DearPyGui
  - 相关
    - ocornut/imgui: Dear ImGui: Bloat-free Graphical User interface for C++ with minimal dependencies
  - 截图
    - 

Python开发iOS/Mac的程序

Q:

脚本-CSDN论坛

> 怎么让苹果手机执行python脚本
>
> 比如淘宝自动签到
>
> 能不能把脚本打包成那种软件之类的形式放在屏幕上

A:

你说的应该是：

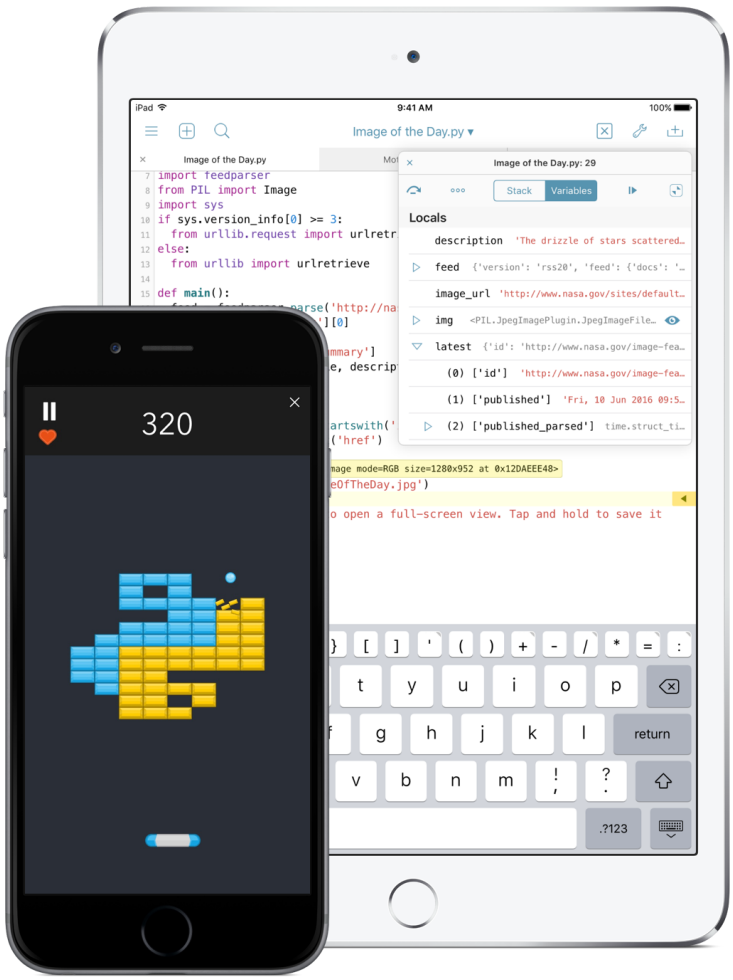
用Python语言写代码，生成可以在iPhone上运行的程序，即打包生成iPhone中的可执行文件，即用Python开发iOS程序

目前能找到的有：`beeware`

- beeware
  - 官网
    - Write once. Deploy everywhere.— BeeWare
      - https://beeware.org
  - 文档入口
    - How to use BeeWare— BeeWare
      - https://beeware.org/project/using/
    - BeeWare — BeeWare 0.3.0 documentation
      - https://docs.beeware.org/en/latest/
    - 关于针对iOS的打包
      - Tutorial 5 - Taking it mobile: iOS — BeeWare 0.3.0 documentation
        - https://docs.beeware.org/en/latest/tutorial/tutorial-5/iOS.html
  - Github
    - 新：Python-Apple-support
      - beeware/Python-Apple-support: A meta-package for building a version of Python that can be embedded into a macOS, iOS, tvOS or watchOS project.
        - https://github.com/beeware/Python-Apple-support
    - 旧=已废弃：Python-iOS-support
      - pybee/Python-iOS-support: A meta-package for building a version of Python that can be embedded into an iOS project.
        - https://github.com/pybee/Python-iOS-support

另外，其他相关的有：

- 用Python开发可以在桌面端Mac运行的程序（app/pkg/dmg等）
  - 可以用Python写代码
  - 再用PyInstaller打包
    - PyInstaller *

- 在iPhone（或iPad）中写Python代码=做Python开发
  - Pythonista for iOS
    - http://omz-software.com/pythonista/
  - 截图
    - 

SSH

- Paramiko
  - 概述
    - Paramiko is a Python (2.7, 3.4+) implementation of the SSHv2 protocol [1], providing both client and server functionality. While it leverages a Python C extension for low level cryptography (Cryptography), Paramiko itself is a pure Python interface around SSH networking concepts.
  - 主页
    - Welcome to Paramiko! — Paramiko documentation
  - 文档
    - Welcome to Paramiko's documentation! — Paramiko documentation
  - Github
    - paramiko/paramiko: The leading native Python SSHv2 protocol library.

附录

下面列出相关参考资料。

相关教程

其他一些Python相关教程：

- 自己(Crifan Li)写的
  - 适合新手看的
    - Python新手小白常见错误和问题
      - https://book.crifan.com/books/python_newbie_mistakes_questions/website/
  - 适合平时开发参考和直接借鉴（拿来直接用）的
    - Python常用代码段
      - https://book.crifan.com/books/python_common_code_snippet/website/
  - 其他Python入门的基础教程
    - python初级教程：入门详解
      - https://www.crifan.com/files/doc/docbook/python_beginner_tutorial/release/html/python_beginner_tutorial.html
- 别人写的
  - 廖雪峰的
    - 新 `Python 3`
      - Python教程 - 廖雪峰的官方网站
    - 旧 `Python 2`
      - Python 2.7教程 - 廖雪峰的官方网站
- Python官网的
  - Python 教程 — Python 3 文档
    - https://docs.python.org/zh-cn/3/tutorial/index.html
      - 注：这个是官网最新的，20210406 最新版已更新到 `Python 3.9.4`
- 其他相关
  - 目录 Full Stack Python 简体中文网站

其他旧Python教程

我(Crifan Li)之前写的，Python相关教程，供需要的参考：

- python中级教程：开发总结
- Python语言总结

参考资料

- 【已解决】Python中的hasattr的含义及实际用法举例
- 【详解】Python 3中字符串的替换str.replace
- 【整理Book】保存数据的仓库：数据库
- 【整理Book】Python心得：操作CSV和Excel
- 【已解决】Python中删除字典dict中的键值
- 【已解决】把Celery+Redis集成到在线Flask中且用supervisor去管理后台服务
- 【已解决】把Python3的pipenv的Flask部署到CentOS服务器上
- 【已解决】Fabric的三个版本fabric1、fabric2和fabric3的区别
-
- 【已解决】python中的dict被copy后仍会被修改
- 轻量但强大的Python框架：Flask
- 【记录】试试Mac中Python自带的IDE工具IDLE的效果
- 【已解决】选择好的Flask的REST API的框架
- 【已解决】Flask中获取REST API接口的传递进来的参数
- 【已解决】CentOS服务器中搭建Python的Flask的REST API
- 【已解决】把Python3的pipenv的Flask部署到CentOS服务器上
- 【已解决】CentOS中如何查看Python的site-packages位置
- Python打包利器：PyInstaller
- 【教程】详解Python中代码缩进（Indent）：影响代码的内在逻辑关系和执行结果 – 在路上
- 【记录】使用Python的IDE：PyScripter – 在路上
- 编辑器和IDE总结
- 【整理】各种Python的IDE(集成开发环境)的总结和对比 – 在路上
- 4.1.3.2. 目前常见的一些Python的IDE - - python初级教程：入门详解
- 调试Python · 史上最好用的编辑器：VSCode
- 【已解决】mac中PyInstaller打包后的二进制文件在electron-builder打包后app中有些无法通过child_process的execFile运行
- 【未解决】pyinstaller打包运行失败：failed to execute script **main**
- ［已解决］pyinstaller打包exe运行出错：fatal error returned -1
- 【记录】用PyInstaller把Python代码打包成单个独立的exe可执行文件 – 在路上
- 【整理】手机上能看的最新的python学习资料
- 【整理】python中的with的含义和用法
- 【整理】Python 特殊方法 魔术方法
- 【已解决】Python中直接return数组和yield的区别
- 【已解决】Python中用正则re去搜索分组的集合
- 【记录】升级宝塔Linux面板
- 【已解决】Python中继承父类如何重写**init**以自定义初始化
- 【已解决】python中把dict的json输出到文件且带缩进和不要unicode的\uxxxx
- 【已解决】Python中实现dict的递归的合并更新
- 【已解决】Python中两个星号**参数去传递给函数出错：SyntaxError invalid syntax
- 【已解决】Python中使用相对路径导入库函数
- 【已解决】Python中递归import导入：ImportError: cannot import name

- 【已解决】python中的相对路径导入库失败No module named
- 【已解决】gunicorn运行Flask的app出错：gunicorn.errors.HaltServer HaltServer Worker failed to boot 3
- 【已解决】gunicorn运行Flask的app但访问出错：org.apache.http.conn.HttpHostConnectException Connect to failed Operation timed out
- 【已解决】把Celery+Redis集成到在线Flask中且用supervisor去管理后台服务
- 【已解决】部署Flask到生产环境服务器上的基本逻辑和大概步骤流程
- 【已解决】部署Flask的WSGI的方式的选择
-
- 谁是2020年最强Python库？年度Top10出炉 - 知乎
- 或许，这是最强大的一款Python GUI工具 - 知乎
- Python GUIs with DearPyGui. A look and guide at a new GUI for your… | by Jonathan Hoffstadt | Aug, 2020 | ITNEXT
- 在iOS中运行Python | ColdCode
- iOS 工程中调用Python方法 - 简书
- 使用Python开发iOS程序 - Forkong - SegmentFault 思否
- python IDE 新手用什么比较好？ - 知乎
- 初学python，pycharm和Spyder哪个好？ - 知乎
- python - What does * *(double star/asterisk) and* (star/asterisk) do for parameters? - Stack Overflow
- Python打包exe的王炸-Nuitka - 知乎
- 整理的一些Python学习资料 - 简书
- Learn Python in Y Minutes
- 10 Useful Tools and Libraries for Programmer and IT Professionals
- Jetbrains发布2019开发者生态报告：Java最主流，Go最有前途
- 干货 | Python后台开发的高并发场景优化解决方案
- Python传奇：30年崛起之路
- 完整中文版 | 2018 Python官方年度报告：关于 Python 的趋势都在这了 - 知乎
- How does Julia compare to Python? - Quora
- Python工程师-CSDN学院
- 社区 — The Hitchhiker's Guide to Python
- PEP 8 -- Style Guide for Python Code | Python.org
- PEP 20 -- The Zen of Python | Python.org
- 吉多·范罗苏姆 - 维基百科，自由的百科全书
- BDFL - Python Wiki
- 社区 — The Hitchhiker's Guide to Python
- Learn Python in Y Minutes
- Python-Guide-CN/style.rst at master · Prodesire/Python-Guide-CN
- awesome-python-cn/README.md at master · jobbole/awesome-python-cn
- 还在用 Python 2.x？Python 3.7.0 正式发布！
- The History of Python: Introduction and Overview
- Python语言在企业级应用上的十大谬误 | 程序师
- Prodesire/Python-Guide-CN: Python最佳实践指南
- “Python 3永远不可能出现在Facebook”，4年后：真香
- 用 Python 开发一个企业级的监控平台 - CSDN博客
- 究竟哪个版本的Python是最快的?
- Python 笔记一：简单入门及点评 - 简书

- 介绍Python2和Python3的兼容库six | 卡瓦邦噶!
- Six: Python 2 and 3 Compatibility Library — six 1.10.0 documentation
- Python 2-3 兼容库 six
- Python3之six库知识扫盲 - CSDN博客
- 写程序怎么做到Python2与Python3兼容？ - 知乎
- Python 2 还是 3？ - Full Stack Python 简体中文网站
- Scrapy on the Road to Python 3 Support – The Scrapinghub Blog
- Python 3 Readiness - Python 3 support table for most popular Python packages
- 监控 - Full Stack Python 简体中文网站
- Python性能监控工具 | Python性能测试工具 | Python监控系统 – OneAPM
- Python - Fabric简介 - Anliven - 博客园
- 使用 Fabric 部署 — Flask 0.10.1 文档
- Python远程部署利器Fabric详解 - Python - 伯乐在线
- fabric实现远程操作和部署 - Python - 伯乐在线
- Python模块学习 - fabric - 一只小小的寄居蟹 - 博客园
-