

Python语言程序设计

第8章 程序设计方法学





前课复习

数字类型及操作

- 整数类型的无限范围及4种进制表示
- 浮点数类型的近似无限范围、小尾数及科学计数法
- +、-、*、/、//、%、**、二元增强赋值操作符
- abs()、divmod()、pow()、round()、max()、min()
- int()、float()、complex()



字符串类型及操作

- 正向递增序号、反向递减序号、<字符串>[M:N:K]
- +、*、len()、str()、hex()、oct()、ord()、chr()
- .lower()、.upper()、.split()、.count()、.replace()
- .center()、.strip()、.join()、.format()格式化



程序的分支结构

- 单分支 *if* 二分支 *if-else* 及紧凑形式
- 多分支 *if-elif-else* 及条件之间关系
- *not and or > >= == <= < !=*
- 异常处理 *try-except-else-finally*



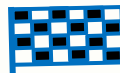
程序的循环结构

- *for...in* 遍历循环: 计数、字符串、列表、文件...
- *while* 无限循环
- *continue* 和 *break* 保留字: 退出当前循环层次
- 循环 *else* 的高级用法: 与 *break* 有关



函数的定义与使用

- 使用保留字`def`定义函数，`lambda`定义匿名函数
- 可选参数(赋初值)、可变参数(*b)、名称传递
- 保留字`return`可以返回任意多个结果
- 保留字`global`声明使用全局变量，一些隐式规则



代码复用与函数递归

- 模块化设计：松耦合、紧耦合
- 函数递归的2个特征：基例和链条
- 函数递归的实现：函数 + 分支结构



集合类型及操作

- 集合使用{}和set()函数创建
- 集合间操作：交(&)、并(|)、差(-)、补(^)、比较(>=<)
- 集合类型方法：.add()、.discard()、.pop()等
- 集合类型主要应用于：包含关系比较、数据去重



序列类型及操作

- 序列是基类类型，扩展类型包括：字符串、元组和列表
- 元组用()和tuple()创建，列表用[]和set()创建
- 元组操作与序列操作基本相同
- 列表操作在序列操作基础上，增加了更多的灵活性



字典类型及操作

- 映射关系采用键值对表达
- 字典类型使用{}和dict()创建，键值对之间用:分隔
- d[key] 方式既可以索引，也可以赋值
- 字典类型有一批操作方法和函数，最重要的是.get()



文件的使用

- 文件的使用方式：打开-操作-关闭
- 文本文件&二进制文件，`open(,)`和`.close()`
- 文件内容的读取：`.read()` `.readline()` `.readlines()`
- 数据的文件写入：`.write()` `.writelines()` `.seek()`



一维数据的格式化和处理

- 数据的维度：一维、二维、多维、高维
- 一维数据的表示：列表类型(有序)和集合类型(无序)
- 一维数据的存储：空格分隔、逗号分隔、特殊符号分隔
- 一维数据的处理：字符串方法 `.split()` 和 `.join()`



二维数据的格式化和处理

- 二维数据的表示：列表类型，其中每个元素也是一个列表
- CSV格式：逗号分隔表示一维，按行分隔表示二维
- 二维数据的处理：for循环+`.split()`和`.join()`





本课概要

第8章 程序设计方法学



- 8.1 实例13: 体育竞技分析
- 8.2 Python程序设计思维
- 8.3 Python第三方库安装
- 8.4 模块7: os库的基本使用
- 8.5 实例14: 第三方库自动安装脚本



第8章 程序设计方法学

方法论

- 理解并掌握Python程序设计思维



实践能力

- 学会编写更有设计感的程序



Python语言程序设计

8.1 实例13: 体育竞技分析





"体育竞技分析"问题分析

问题分析

体育竞技分析



高手过招，胜负只在毫厘之间
因此，就要挖掘并解释竞技背后的规律和现象-体育竞技分析

问题分析

体育竞技分析

- **需求：毫厘是多少？如何科学分析体育竞技比赛？**
- **输入：球员的水平**
- **输出：可预测的比赛成绩**

问题分析

体育竞技分析：模拟N场比赛

- 计算思维：抽象 + 自动化
- 模拟：抽象比赛过程 + 自动化执行N场比赛
- 当N越大时，比赛结果分析会越科学

问题分析

比赛规则

- **双人击球比赛：A & B，回合制，5局3胜**
- **开始时一方先发球，直至判分，接下来胜者发球**
- **球员只能在发球局得分，15分胜一局**

怎么来设计一段程序模拟这个过程呢



自顶向下(设计)

解决复杂问题的有效方法

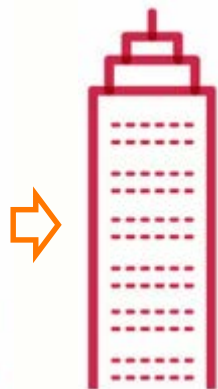
- 将一个总问题表达为若干个小问题组成的形式
- 使用同样方法进一步分解小问题
- 直至，小问题可以用计算机简单明了的解决

程序应采用自顶向下的设计方法，和自底向上的执行方法

自顶向下(设计)

解决复杂问题的有效方法

改善
居住条件



组织
设计和施工



程序应采用自顶向下的设计方法，和自底向上的执行方法

自底向上(执行)

逐步组建复杂系统的有效测试方法

- **分单元测试，逐步组装**
- **按照自顶向下相反的路径操作**
- **直至，系统各部分以组装的思路都经过测试和验证**

先运行和测试每一个基本函数，再测试由基础函数组成的整体函数，有助于定位错误

程序应采用自顶向下的设计方法，和自底向上的执行方法

自底向上(执行)

逐步组建复杂系统的有效测试方法



程序应采用自顶向下的设计方法，和自底向上的执行方法



"体育竞技分析"实例讲解

体育竞技分析

程序总体框架及步骤

- 步骤1：打印程序的介绍性信息
- 步骤2：获得程序运行参数：proA, proB, n
- 步骤3：利用球员A和B的能力值，模拟n局比赛
- 步骤4：输出球员A和B获胜比赛的场次及概率

采用自顶向下的设计方法，最重要的是顶层设计

体育竞技分析

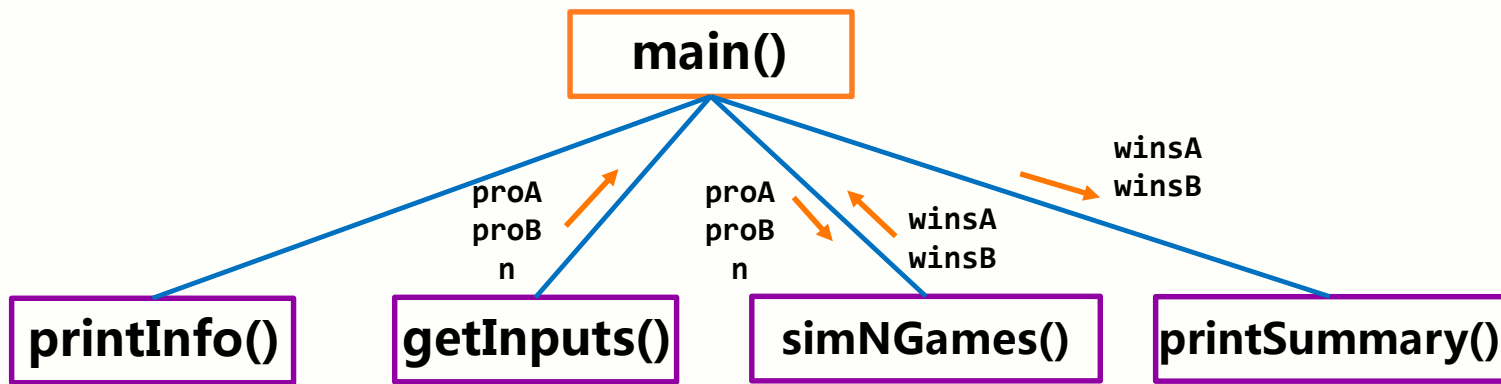
程序总体框架及步骤

- **步骤1：打印程序的介绍性信息** - **printInfo()**
- **步骤2：获得程序运行参数：proA, proB, n** - **getInputs()**
- **步骤3：利用球员A和B的能力值，模拟n局比赛** - **simNGames()**
- **步骤4：输出球员A和B获胜比赛的场次及概率** - **printSummary()**

原问题被划分了4个独立的函数，这样就可以只专心考虑程序的结构设计而不必关心具体细节

体育竞技分析

第一阶段：程序总体框架及步骤



经过顶层设计，main()函数成为顶层结构，其中每层按照从左至右的顺序执行
每一个函数用一个矩形表示，连接两个矩形的线表示上面函数对下面函数的调用关系
在信息流方面，箭头和注释表示函数之间的输入和输出

体育竞技分析

第一阶段

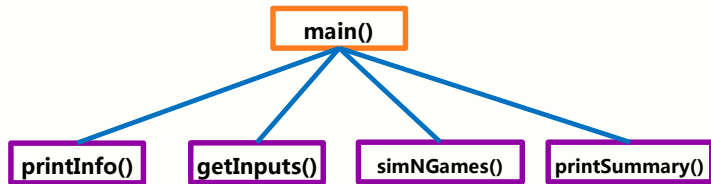
def main():

 printIntro()

 probA, probB, n = getInputs()

 winsA, winsB = simNGames(n, probA, probB)

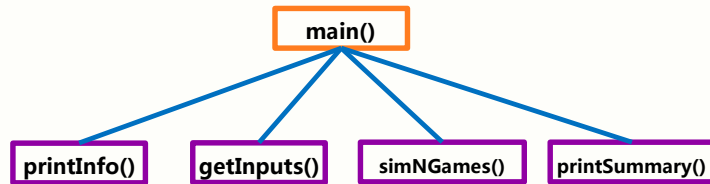
 printSummary(winsA, winsB)



问题的程序框架已经清晰，被划分成4个独立的函数，函数的名称、输入参数和预期返回值都已确定

体育竞技分析

第一阶段



def printIntro():

print("这个程序模拟两个选手A和B的某种竞技比赛")

print("程序运行需要A和B的能力值(以0到1之间的小数表示)")

介绍性内容，提高用户体验

体育竞技分析

第一阶段

```
def getInputs():
```

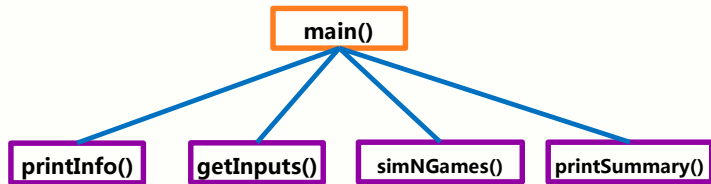
```
    a = eval(input("请输入选手A的能力值(0-1): "))
```

```
    b = eval(input("请输入选手B的能力值(0-1): "))
```

```
    n = eval(input("模拟比赛的场次: "))
```

```
    return a, b, n
```

获得用户的输入，并为主程序返回这些值



体育竞技分析

第一阶段

```
def printSummary(winsA, winsB):
```

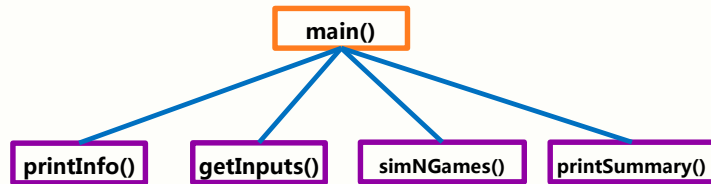
```
    n = winsA + winsB
```

```
    print("竞技分析开始，共模拟{}场比赛".format(n))
```

```
    print("选手A获胜{}场比赛，占比{:0.1%}".format(winsA, winsA/n))
```

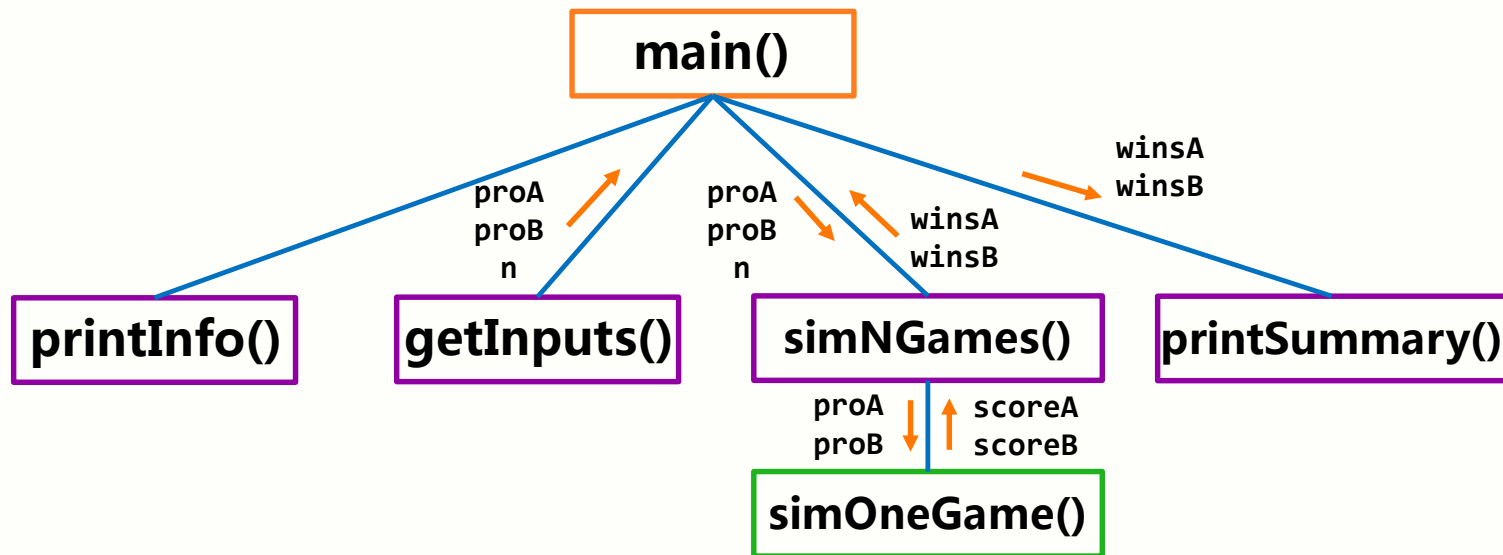
```
    print("选手B获胜{}场比赛，占比{:0.1%}".format(winsB, winsB/n))
```

打印结果，输出获胜比赛场次和概率



体育竞技分析

第二阶段：步骤3 模拟N局比赛

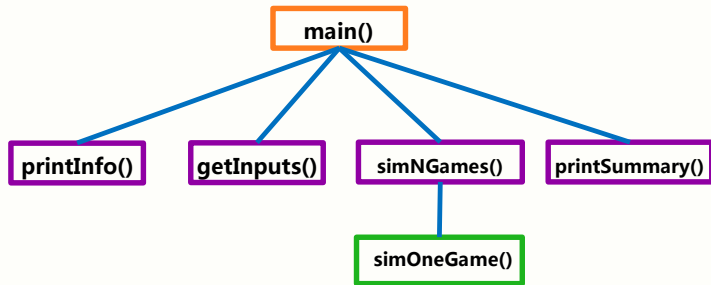


simNGames()是整个程序的核心，思路是模拟n场比赛，并跟踪记录每个球员赢得了多少比赛
simOneGame()函数，用于模拟一场比赛，输入为每个球员的概率，返回为两个球员的最终得分

体育竞技分析

第二阶段

```
def simNGames(n, probA, probB):  
    winsA, winsB = 0, 0  
    for i in range(n):  
        scoreA, scoreB = simOneGame(probA, probB)  
        if scoreA > scoreB:  
            winsA += 1  
        else:  
            winsB += 1  
    return winsA, winsB
```

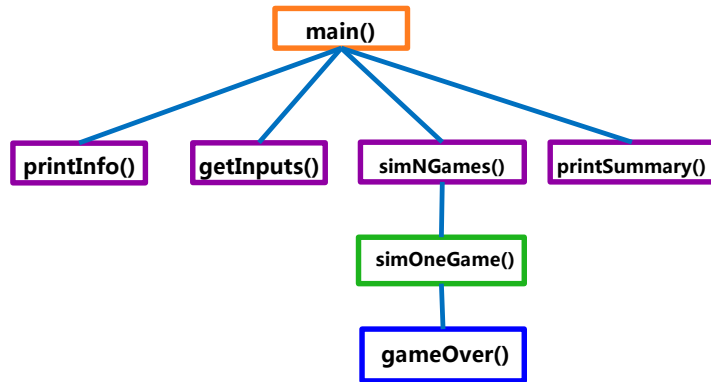


`simNGames()`是整个程序的核心，思路是模拟n场比赛，并跟踪记录每个球员赢得了多少比赛

体育竞技分析

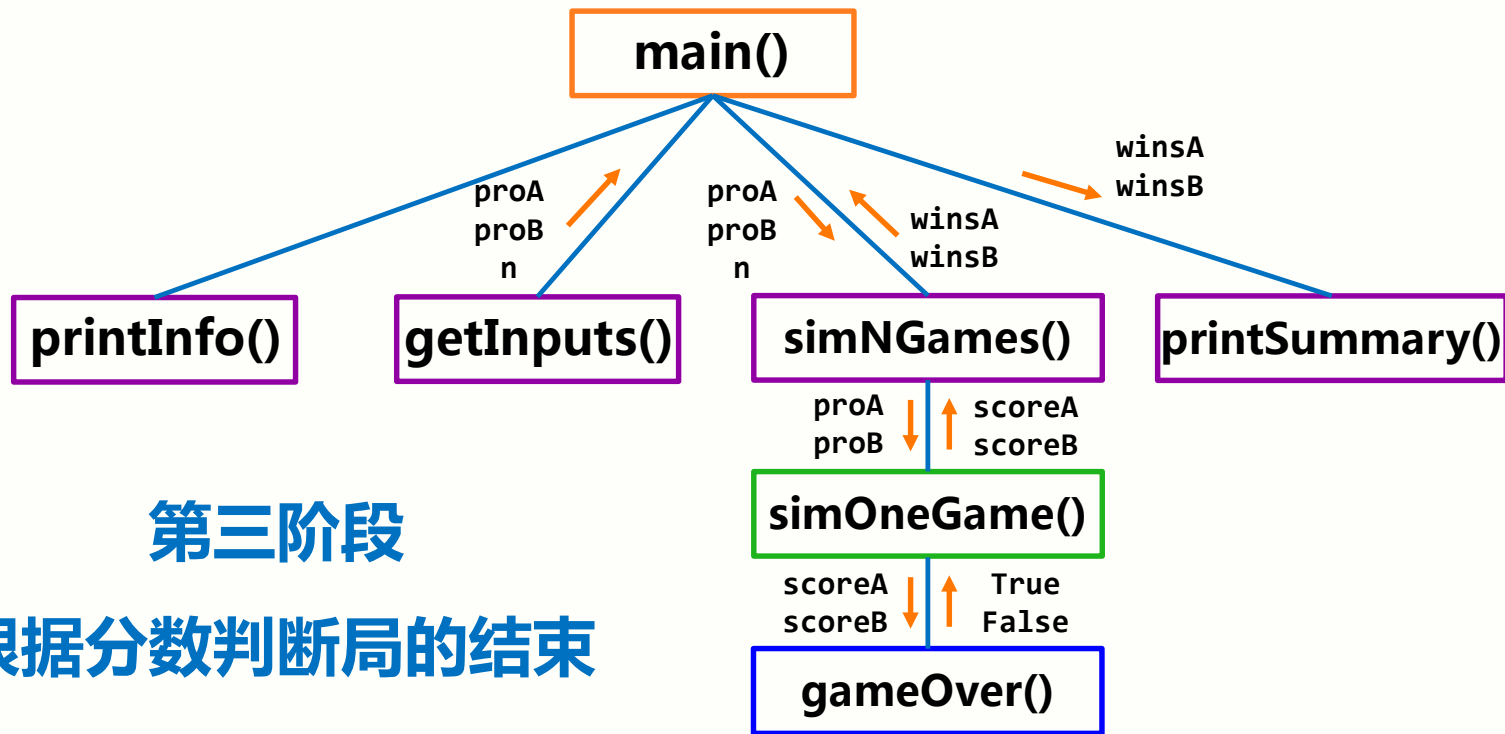
```
def simOneGame(probA, probB):  
    scoreA, scoreB = 0, 0  
    serving = "A"  
    while not gameOver(scoreA, scoreB):  
        if serving == "A":  
            if random() < probA:  
                scoreA += 1  
            else:  
                serving="B"  
        else:  
            if random() < probB:  
                scoreB += 1  
            else:  
                serving="A"  
    return scoreA, scoreB
```

第二阶段



`simOneGame()`函数，用于模拟一场比赛，输入为每个球员的概率，返回为两个球员的最终得分

体育竞技分析

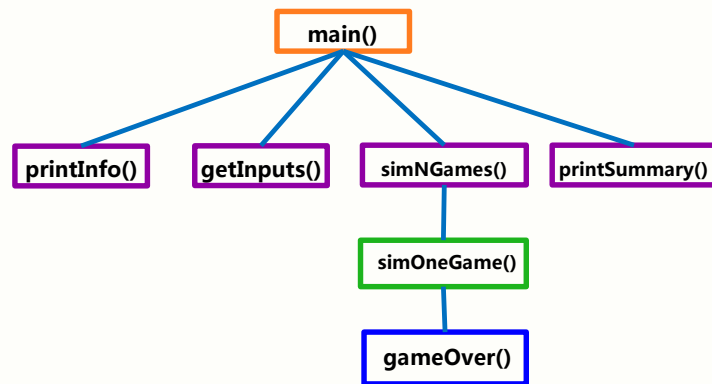


体育竞技分析

```
def simOneGame(probA, probB):  
    scoreA, scoreB = 0, 0  
    serving = "A"  
    while not gameOver(scoreA, scoreB):  
        if serving == "A":  
            if random() < probA:  
                scoreA += 1  
            else:  
                serving="B"  
        else:  
            if random() < probB:  
                scoreB += 1  
            else:  
                serving="A"
```

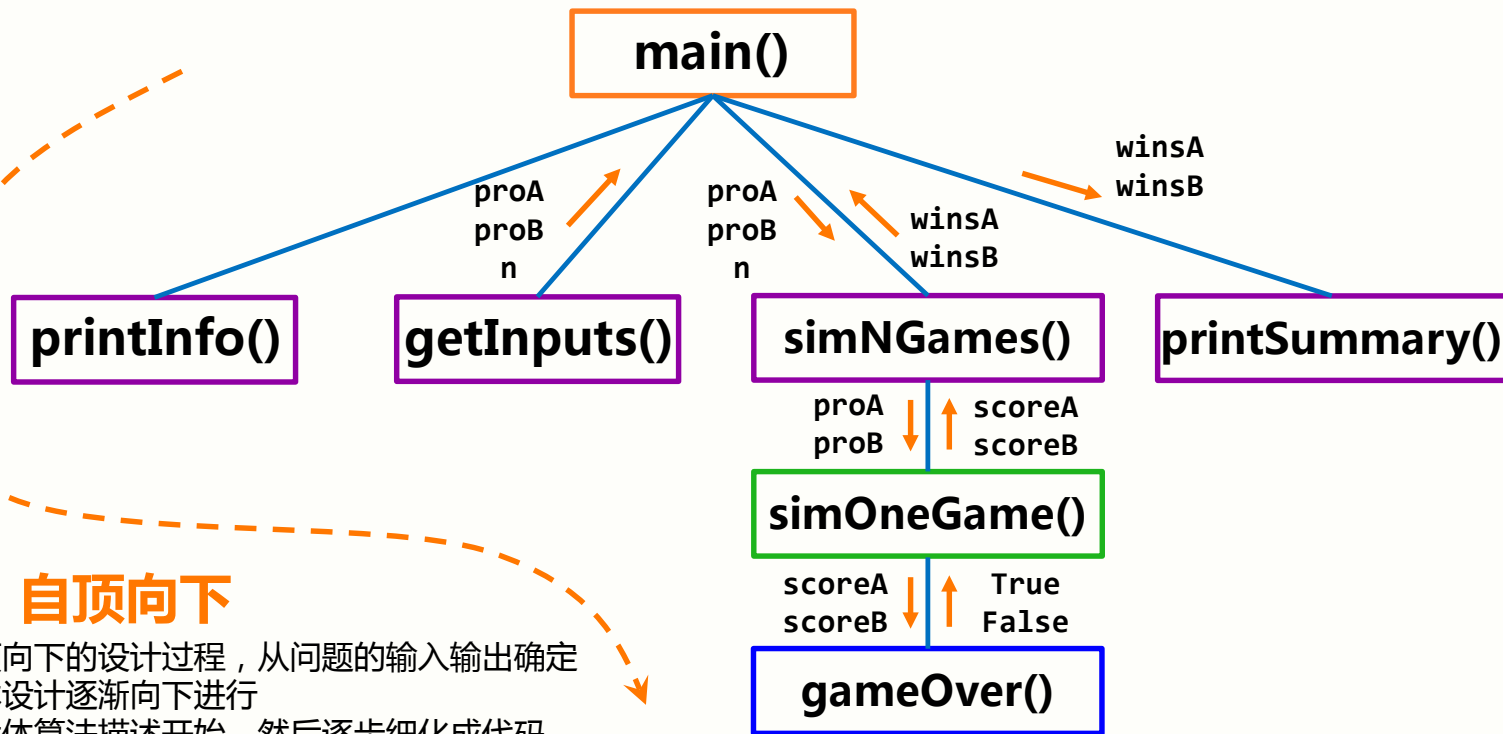
第三阶段

return scoreA, scoreB 设计一个gameOver()函数，用来表示一场比赛结束的条件
对于不同的问题结束条件可能不同，封装该函数有助于简化根据不同规则修改函数的代价，提高代码的可维护性



```
def gameOver(a,b):#到15分停止比赛  
    return a==15 or b==15
```

体育竞技分析



- ✓ 介绍了自顶向下的设计过程，从问题的输入输出确定开始，整体设计逐渐向下进行
- ✓ 每一层以大体算法描述开始，然后逐步细化成代码，细节被函数封装

体育竞技分析

>>>

这个程序模拟两个选手A和B的某种竞技比赛

程序运行需要A和B的能力值（以0到1之间的小数表示）

请输入选手A的能力值(0-1): 0.45

请输入选手B的能力值(0-1): 0.50

模拟比赛的场次: 1000

竞技分析开始，共模拟1000场比赛

选手A获胜365场比赛，占比36.5%

选手B获胜635场比赛，占比63.5%

能力值 : 0.45 v.s. 0.50

获胜数 : 36.5% v.s. 63.5%



"体育竞技分析"举一反三

举一反三

理解自顶向下和自底向上

- 理解自顶向下的设计思维：分而治之
- 理解自底向上的执行思维：模块化集成
- 自顶向下是“系统”思维的简化

自顶向下是一种开发复杂程序最具价值的设计理念和工具，设计过程自然且简单
自顶向下设计通过封装实现抽象，利用了模块化设计的思想

举一反三

应用问题的扩展

- 扩展比赛参数，增加对更多能力对比情况的判断
- 扩展比赛设计，增加对真实比赛结果的预测
- 扩展分析逻辑，反向推理，用胜率推算能力？

Python语言程序设计

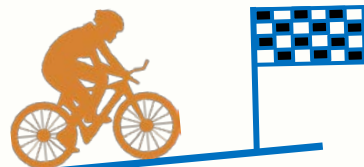
8.2 Python程序设计思维

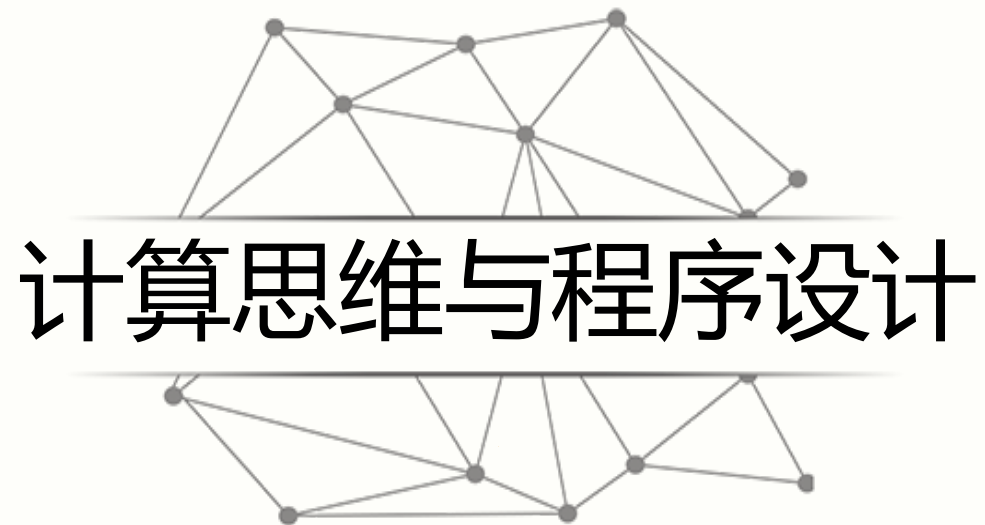


Python程序设计思维



- 计算思维与程序设计
- 计算生态与Python语言
- 用户体验与软件产品
- 基本的程序设计模式





计算思维与程序设计

计算思维

第3种人类思维特征

- **逻辑思维**：推理和演绎，数学为代表， $A \rightarrow B$ $B \rightarrow C$ $A \rightarrow C$
- **实证思维**：实验和验证，物理为代表，引力波 \leftarrow 实验
- **计算思维**：设计和构造，计算机为代表，汉诺塔递归

计算思维

抽象和自动化

- 计算思维：Computational Thinking
- 抽象问题的计算过程，利用计算机自动化求解
- 计算思维是基于计算机的思维方式

这个概念第一次从思维层面阐述了运用计算机科学的基础概念求解问题、设计系统和理解人类行为的过程

计算思维的本质是：抽象和自动化

抽象实际问题的计算特性，利用计算机求解

计算思维

计数求和：计算1-100的计数和

$$S = \frac{(a_1 + a_n)n}{2}$$

```
s = 0
```

```
for i in range(1, 101):
```

```
    s += i
```

逻辑思维

数学家高斯的玩儿法

计算思维

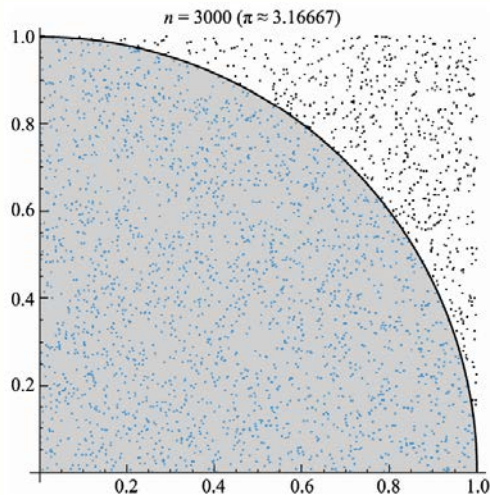
现代人的新玩儿法

计算思维

圆周率的计算

$$\pi = \sum_{k=0}^{\infty} \left[\frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right]$$

逻辑思维



计算思维

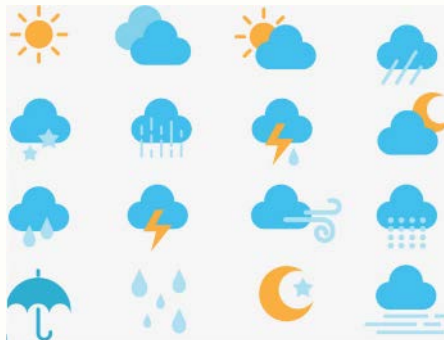
计算思维



实证思维+逻辑思维

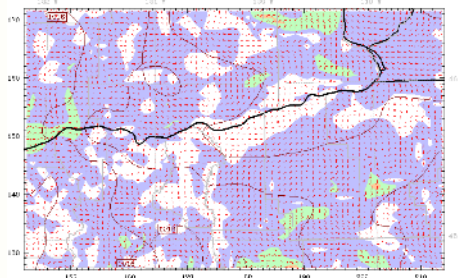
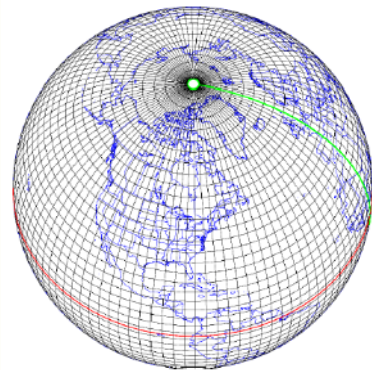
经验
→
猜

天气预报



MM5模型

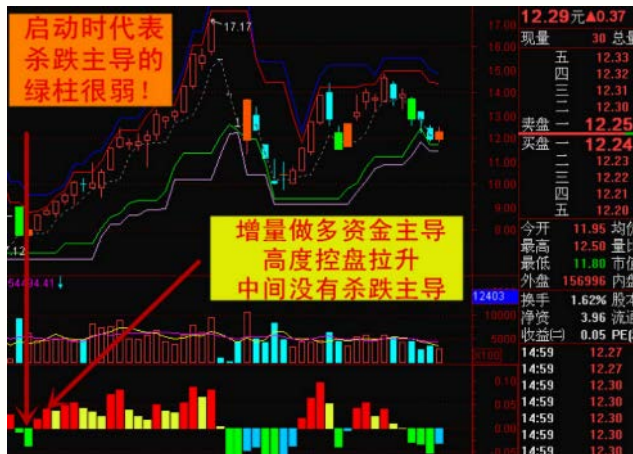
@超算



计算思维

计算思维

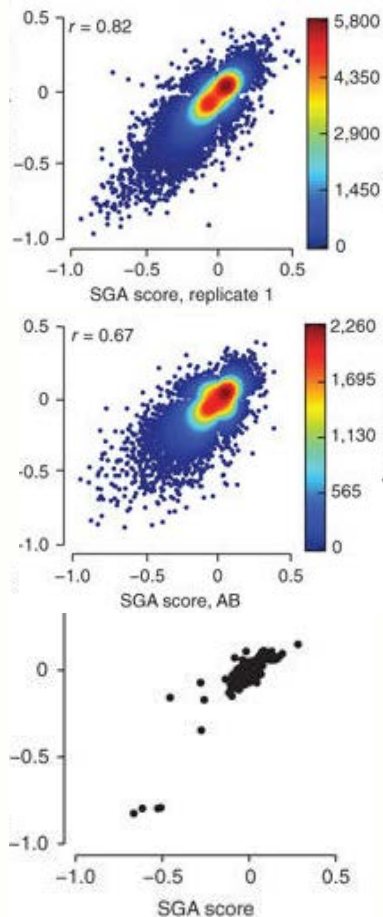
量化分析



实证思维+逻辑思维



计算思维



计算思维

抽象问题的计算过程，利用计算机自动化求解

- 计算思维基于计算机强大的算力及海量数据
- 抽象计算过程，关注设计和构造，而非因果
- 计算机程序设计是实现计算思维的主要手段



计算思维与程序设计

编程是将计算思维变成现实的手段



在程序设计范畴，计算思维主要反映在理解问题的计算特性、将计算特性抽象为计算问题、通过程序设计语言实现问题的自动化求解等

计算思维 真的很有用...



计算生态与Python语言

计算生态

从开源运动说起...



- 1983, Richard Stallman启动GNU项目
- 1989, GNU通用许可协议诞生

自由软件时代到来

计算生态

从开源运动说起...



- 1991, Linus Torvalds发布了Linux内核
- 1998, 网景浏览器开源，产生了Mozilla

开源生态逐步建立

计算生态

从开源运动说起...



V.S.



- 1983, Richard Stallman

大教堂模式

- 1991, Linus Torvalds

集市模式

计算生态

开源思想深入演化和发展，形成了计算生态

计算生态以开源项目为组织形式，充分利

用“共识原则”和“社会利他”组织人员，在竞争发展、相互依存和迅速更迭中完成信息技术的更新换代，形成了技术的自我演化路径。



计算生态

没有顶层设计、以功能为单位、具备三个特点



- 竞争发展
- 相互依存
- 迅速更迭



计算生态与Python语言

- 以开源项目为代表的大量第三方库

Python语言提供 >15万个第三方库

- 库的建设经过野蛮生长和自然选择

同一个功能，Python语言2个以上第三方库

计算生态与Python语言

- 库之间相互关联使用，依存发展

Python库间广泛联系，逐级封装

- 社区庞大，新技术更迭迅速

AlphaGo深度学习算法采用Python语言开源

计算生态的价值

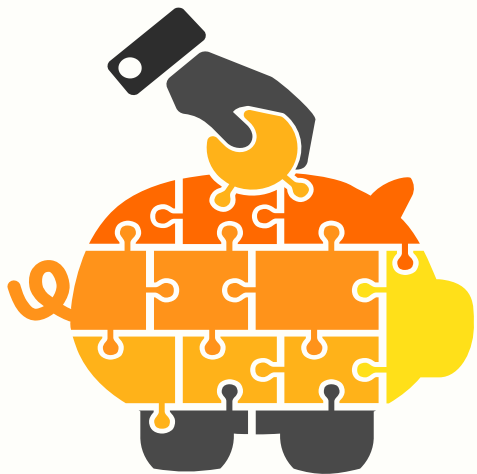
创新：跟随创新、集成创新、原始创新



- **加速科技类应用创新的重要支撑**
- **发展科技产品商业价值的重要模式**
- **国家科技体系安全和稳固的基础**

计算生态的运用

刀耕火种 -> 站在巨人的肩膀上



- 编程的起点不是**算法**而是**系统**
- 编程如同搭积木，利用计算生态为主要模式
- 编程的目标是**快速**解决问题

理解和运用计算生态



用户体验与及软件产品

用户体验

实现功能 -> 关注体验

- **用户体验指用户对产品建立的主观感受和认识**
- **关心功能实现，更要关心用户体验，才能做出好产品**
- **编程只是手段，不是目的，程序最终为人类服务**

提高用户体验的方法

方法1：进度展示

- 如果程序需要计算时间，可能产生等待，请增加进度展示
- 如果程序有若干步骤，需要提示用户，请增加进度展示
- 如果程序可能存在大量次数的循环，请增加进度展示

提高用户体验的方法

方法2：异常处理

- 当获得用户输入，对合规性需要检查，需要异常处理
- 当读写文件时，对结果进行判断，需要异常处理
- 当进行输入输出时，对运算结果进行判断，需要异常处理

提高用户体验的方法

其他类方法

- **打印输出**：特定位置，输出程序运行的过程信息
- **日志文件**：对程序异常及用户使用进行定期记录
- **帮助信息**：给用户多种方式提供帮助信息

软件程序 -> 软件产品

用户体验是程序到产品的关键环节



基本的程序设计模式

基本的程序设计模式

从IPO开始...

- **I : Input 输入，程序的输入**
- **P : Process 处理，程序的主要逻辑**
- **O : Output 输出，程序的输出**

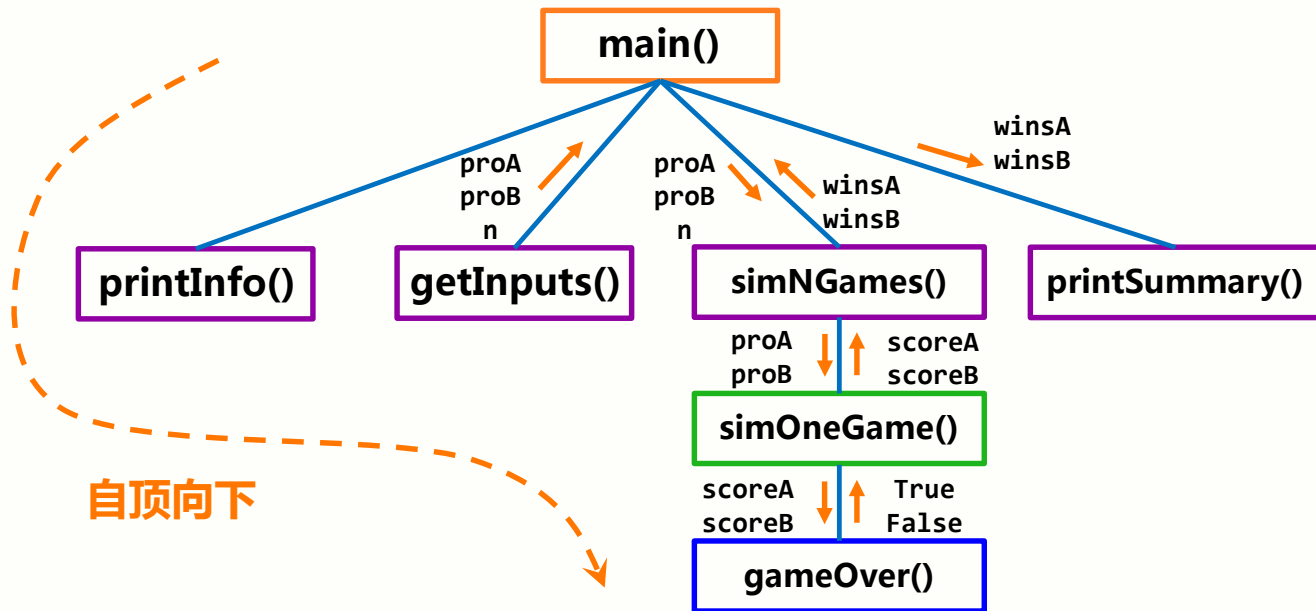
基本的程序设计模式

从IPO开始...

- **确定IPO：明确计算部分及功能边界**
- **编写程序：将计算求解的设计变成现实**
- **调试程序：确保程序按照正确逻辑能够正确运行**

基本的程序设计模式

自顶向下设计



基本的程序设计模式

模块化设计

- 通过函数或对象封装将程序划分为模块及模块间的表达
- 具体包括：主程序、子程序和子程序间关系
- 分而治之：一种分而治之、分层抽象、体系化的设计思想

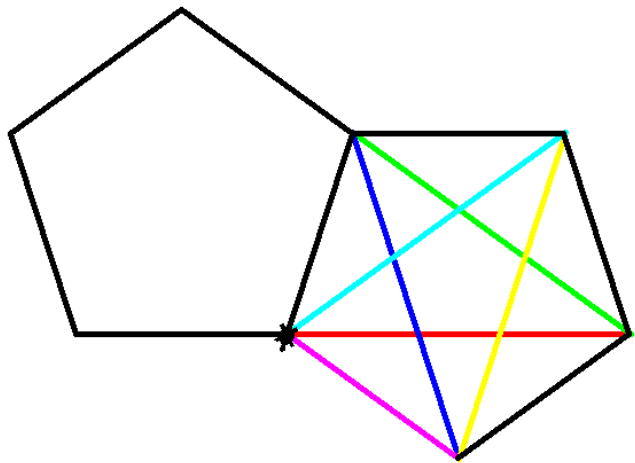
基本的程序设计模式

模块化设计

- **紧耦合：两个部分之间交流很多，无法独立存在**
- **松耦合：两个部分之间交流较少，可以独立存在**
- **模块内部紧耦合、模块之间松耦合**

基本的程序设计模式

配置化设计



程序引擎



配置文件

基本的程序设计模式

配置化设计

- **引擎+配置**：程序执行和配置分离，将可选参数配置化
- 将程序开发变成配置文件编写，扩展功能而不修改程序
- 关键在于接口设计，清晰明了、灵活可扩展

应用开发的四个步骤

从应用需求到软件产品

- 1 产品定义

- 3 设计与实现

- 2 系统架构

- 4 用户体验



应用开发的四个步骤

从应用需求到软件产品

- 1 **产品定义**：对应用需求充分理解和明确定义
产品定义，而不仅是功能定义，要考虑商业模式
- 2 **系统架构**：以系统方式思考产品的技术实现
系统架构，关注数据流、模块化、体系架构

应用开发的四个步骤

从应用需求到软件产品

- 3 **设计与实现**：结合架构完成关键设计及系统实现
结合可扩展性、灵活性等进行设计优化
- 4 **用户体验**：从用户角度思考应用效果
用户至上，体验优先，以用户为中心



单元小结

Python程序设计思维

- **计算思维：抽象计算过程和自动化执行**
- **计算生态：竞争发展、相互依存、快速更迭**
- **用户体验：进度展示、异常处理等**
- **IPO、自顶向下、模块化、配置化、应用开发的四个步骤**

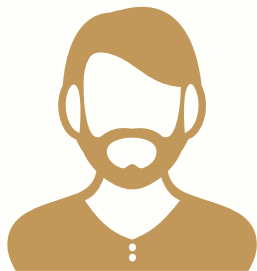


Python语言程序设计

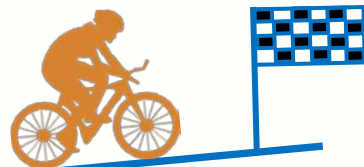
8.3 Python第三方库安装



Python第三方库安装



- 看见更大的Python世界
- 第三方库的pip安装方法
- 第三方库的集成安装方法
- 第三方库的文件安装方法

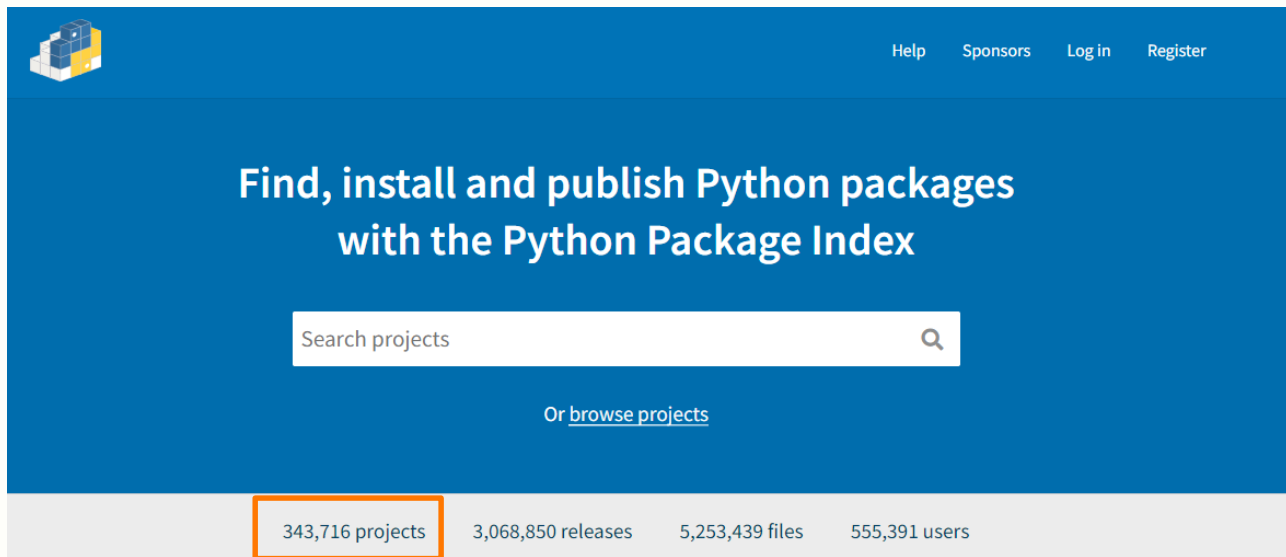




看见更大的Python世界

Python社区

> 34万个第三方库 <https://pypi.org/>



Python官方网站提供了第三方库索引功能（ the python package index,pypi ） , 有26万多
这些函数库覆盖信息领域技术所有技术方向

Python社区

PyPI

- PyPI: **Py**thon **P**ackage **I**ndex
- PSF维护着展示全球Python计算生态的主站
- 学会检索并利用PyPI , 找到合适的第三方库开发程序

PSF: python software foundation

Python社区

实例：开发与区块链相关的程序

- 第1步：在pypi.org搜索 blockchain
- 第2步：挑选适合开发目标的第三方库作为基础
- 第3步：完成自己需要的功能

Python社区

实例：开发与区块链相关的程序

The screenshot shows the PyPI search results for the keyword 'blockchain'. The search bar at the top contains the text 'blockchain'. Below the search bar, the results are filtered by 'classifier'. The search results show 606 projects for 'blockchain'. The results are ordered by 'Relevance'. The first five results are highlighted with a dashed orange box:

Project Name	Description	Release Date
blockchain 1.4.4	Blockchain API library (v1)	May 29, 2018
blockchain-certificates 2.0.2	Create pdf certificate files and issue on the blockchain!	Aug 27, 2020
estudos-blockchain 1.0	Estudos simples sobre blockchain	Sep 24, 2019
chia-blockchain 0.1	Chia proof of space plotting, proving, and verifying (wraps C++)	Mar 31, 2020
Clique-blockchain 0.2.3	Python API for Clique block chains; ID chains and auth chains included.	Feb 6, 2017

On the left side of the search results, there is a 'Filter by classifier' section with the following filters:

- Framework
- Topic
- Development Status
- License
- Programming Language
- Operating System
- Environment
- Intended Audience
- Natural Language
- Typing

安装Python第三方库

三种方法

- **方法1(主要方法): 使用pip命令**
- **方法2: 集成安装方法**
- **方法3: 文件安装方法**



第三方库的pip安装方法

pip安装方法

D:\>pip -h

使用pip安装工具（命令行下执行）

Usage:

pip <command> [options]

Commands:

install	Install packages.
download	Download packages.
uninstall	Uninstall packages.
freeze	Output installed packages in requirements format.
list	List installed packages.
show	Show information about installed packages.
check	Verify installed packages have compatible dependencies.
search	Search PyPI for packages.
wheel	Build wheels from your requirements.
help	Show help for commands.



Windows



Mac OS



Linux

- ✓ pip是python最常用且最高效的第三方库安装工具，是官方提供并维护
- ✓ 是python内置命令，需要通过命令行执行
- ✓ pip -h 命令将列出pip常用的子命令

pip安装方法

常用的pip命令

D:\>pip install <第三方库名>

- 安装指定的第三方库

pip安装方法

常用的pip命令

D:\>pip install -U <第三方库名>

- 使用-U标签**更新**已安装的指定第三方库

pip安装方法

常用的pip命令

D:\>pip uninstall <第三方库名>

- 卸载指定的第三方库

pip安装方法

常用的pip命令

D:\>pip download <第三方库名>

- 下载但不安装指定的第三方库

pip安装方法

常用的pip命令

D:\>pip show <第三方库名>

- 列出某个指定第三方库的详细信息

pip安装方法

常用的pip命令

D:\>pip search <关键词>

- 根据关键词在名称和介绍中搜索第三方库

pip安装方法

pip search blockchain

命令提示符

```
C:\Users\Tian Song>pip search blockchain
blockchain (1.4.0) - Blockchain API library (v1)
easy-blockchain (0.1.6) - A blockchain for human
blockchain-parser (0.1.4) - Bitcoin blockchain parser
blockchain-certificates (0.10.4) - Create pdf certificate files and issue
                                on the blockchain!
                                - Issues blockchain certificates using
                                the Bitcoin blockchain
cert-issuer (2.0.12) - A blockchain linker to help understand
                    hash functions in blockchain
justblockchain (1.0.0) - A library for constructing virtual
                    blockchains within a cryptocurrency's
                    blockchain
virtualchain (0.18.0.1) - Digital Tokens on the Bitcoin
                    Blockchain
assemblycoins (0.1.2) - BigchainDB: A Scalable Blockchain
                    Database
BigchainDB (1.3.0) - Blockchain Auth Library
blockchainauth (0.2.0) - Blockchain-inspired datastore
blockrecord (0.1.0) - Bitcoin blockchain parser
blocktools (0.1.3) - Graphene blockchains management tools
bts_tools (0.5.3) - Catena is blockchain as a service.
catena (0.0.1.dev8) - tools for working with blockchain
                    certificates
cert-schema_pastday (2.0b1) - A library for retrieving blockchain
                    certificates
cert-store (2.0.5) - creates blockchain certificates
cert-tools (2.0.9) - Verifies blockchain certificates
cert-verifier (2.0.12) - Blockchain and Cryptonote Framework
Cryptonote (0.0.5)
```

pip安装方法

常用的pip命令

```
D:\>pip list
```

- 列出当前系统已经安装的第三方库

pip安装方法

主要方法，适合99%以上情况

- **适合Windows、Mac和Linux等操作系统**
- **未来获取第三方库的方式，目前的主要方式**
- **适合99%以上情况，需要联网安装**

然而，由于一些历史、技术和政策等原因，有一些第三方库暂时无法用pip安装，需要其他的安装方法



第三方库的集成安装方法

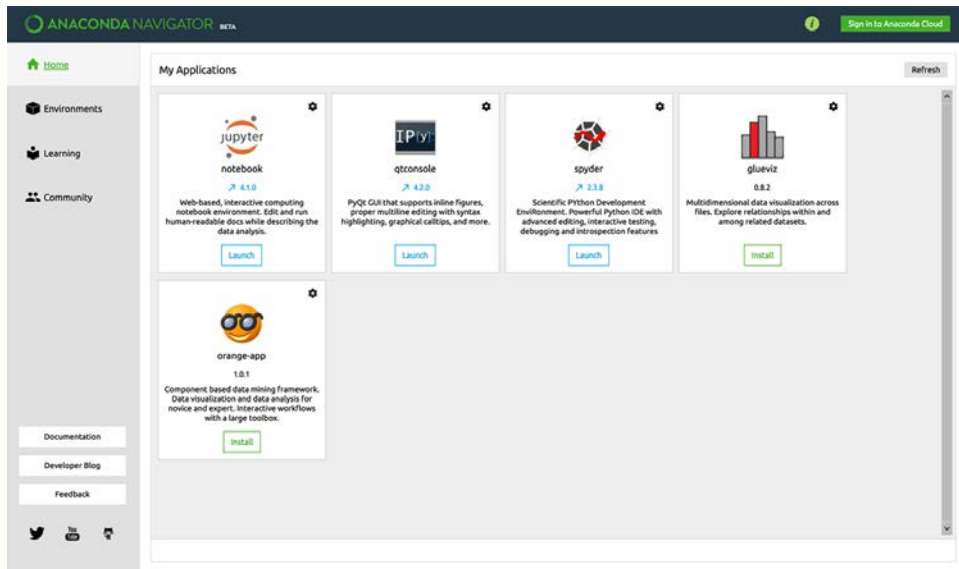
集成安装方法

集成安装：结合特定Python开发工具的批量安装

Anaconda

<https://www.continuum.io>

- 支持近800个第三方库
- 包含多个主流工具
- 适合数据计算领域开发





第三方库的文件安装方法

文件安装方法

为什么有些第三方库用pip可以下载，但无法安装？

- 某些第三方库pip下载后，需要编译再安装
- 如果操作系统没有编译环境，则能下载但不能安装
- 可以直接下载编译后的版本用于安装吗？

文件安装方法

<http://www.lfd.uci.edu/~gohlke/pythonlibs/>

Unofficial Windows Binaries for Python Extension Packages

by Christoph Gohlke, Laboratory for Fluorescence Dynamics, University of California, Irvine.

This page provides 32- and 64-bit Windows binaries of many scientific open-source extension packages for the official [CPython distribution](#) of the [Python](#) programming language.

The files are unofficial (meaning: informal, unrecognized, personal, unsupported, no warranty, no liability, provided "as is") and made available for testing and evaluation purposes.

Most binaries are built from source code found on [PyPI](#) or in the projects public revision control systems. Source code changes, if any, have been submitted to the project maintainers or are included in the packages.

Refer to the documentation of the individual packages for license restrictions and dependencies.

If downloads fail, reload this page, enable JavaScript, disable download managers, disable proxies, clear cache, use Firefox, reduce number and frequency of downloads. Please only download files manually as needed.

Use [pip](#) version 9 or newer to [install the downloaded .whl files](#). This page is not a pip package index.

Many binaries depend on [numpy-1.13+mk1](#) and the Microsoft Visual C++ 2008 ([x64](#), [x86](#), and [SP1](#) for CPython 2.7), Visual C++ 2010 ([x64](#), [x86](#), for CPython 3.4), or the Visual C++ 2017 ([x64](#) or [x86](#) for CPython 3.5, 3.6, and 3.7) redistributable packages.

Install [numpy+mk1](#) before other packages that depend on it.

The binaries are compatible with the most recent official CPython distributions on Windows >=6.0. Chances are they do not work with custom Python distributions included with Blender, Maya, ArcGIS, QGIS, ABAQUS, Cygwin, Pythonxy, Canopy, EPD, Anaconda, WinPython etc. Many binaries are not compatible with Windows XP or Wine.

The packages are ZIP or 7z files, which allows for manual or scripted installation or repackaging of the content.

The files are provided "as is" without warranty or support of any kind. The entire risk as to the quality and performance is with you.

The opinions or statements expressed on this page should not be taken as a position or endorsement of the Laboratory for Fluorescence Dynamics or the University of California.



Windows

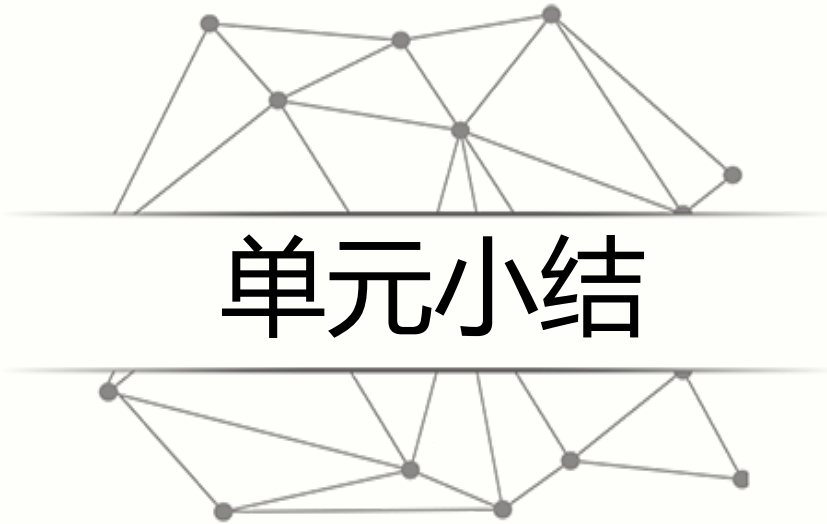
UCI页面

美国加州大学尔湾分校提供了一个页面，帮助python用户获得windows可直接安装的第三方库文件

文件安装方法

实例：安装wordcloud库

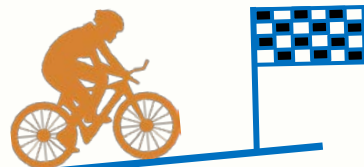
- 步骤1：在UCI页面上搜索wordcloud
- 步骤2：下载对应版本的文件
- 步骤3：使用pip install <文件名>安装



单元小结

Python第三方库安装

- **PyPI : Python Package Index**
- **pip命令的各种用法**
- **Anaconda集成开发工具及安装方法**
- **UCI页面的“补丁”安装方法**



8.4 模块7: os库的使用





os库基本介绍

os库基本介绍

os库提供通用的、基本的操作系统交互功能



Windows



Mac OS



Linux

- **os库是Python标准库，包含几百个函数**
- **常用路径操作、进程管理、环境参数等几类**

os库基本介绍

- **路径操作**：os.path子库，处理文件路径及信息
- **进程管理**：启动系统中其他程序
- **环境参数**：获得系统软硬件信息等环境参数



os库之路径操作

路径操作

os.path子库以path为入口，用于操作和处理文件路径

```
import os.path
```

或

```
import os.path as op
```

路径操作

函数	描述
os.path.abspath(path)	返回path在当前系统中的绝对路径 <pre>>>>os.path.abspath("file.txt") 'C:\\Users\\Tian Song\\Python36-32\\file.txt'</pre>
os.path.normpath(path)	归一化path的表示形式，统一用\\分隔路径 <pre>>>>os.path.normpath("D://PYE//file.txt") 'D:\\PYE\\file.txt'</pre>
os.path.relpath(path)	返回当前程序与文件之间的相对路径 (relative path) <pre>>>>os.path.relpath("C://PYE//file.txt") '..\\..\\..\\..\\..\\..\\..\\..\\..\\PYE\\file.txt'</pre>

路径操作

函数	描述
<code>os.path.dirname(path)</code>	返回path中的目录名称 <pre>>>>os.path.dirname("D://PYE//file.txt") 'D://PYE'</pre>
<code>os.path.basename(path)</code>	返回path中最后的文件名称 <pre>>>>os.path.basename("D://PYE//file.txt") 'file.txt'</pre>
<code>os.path.join(path, *paths)</code>	组合path与paths , 返回一个路径字符串 <pre>>>>os.path.join("D:/", "PYE/file.txt") 'D:/PYE/file.txt'</pre>

路径操作

函数	描述
os.path.exists(path)	判断path对应文件或目录是否存在，返回True或False <pre>>>>os.path.exists("D://PYE//file.txt") False</pre>
os.path.isfile(path)	判断path所对应是否为已存在的文件，返回True或False <pre>>>>os.path.isfile("D://PYE//file.txt") True</pre>
os.path.isdir(path)	判断path所对应是否为已存在的目录，返回True或False <pre>>>>os.path.isdir("D://PYE//file.txt") False</pre>

路径操作

函数	描述
os.path.getatime(path)	返回path对应文件或目录上一次的访问时间 <pre>>>>os.path.getatime("D:/PYE/file.txt") 1518356633.7551725</pre>
os.path.getmtime(path)	返回path对应文件或目录最近一次的修改时间 <pre>>>>os.path.getmtime("D:/PYE/file.txt") 1518356633.7551725</pre>
os.path.getctime(path)	返回path对应文件或目录的创建时间 <pre>>>time.ctime(os.path.getctime("D:/PYE/file.txt")) 'Sun Feb 11 21:43:53 2018'</pre>

路径操作

函数	描述
<code>os.path.getsize(path)</code>	<p>返回path对应文件的大小，以字节为单位</p> <pre>>>>os.path.getsize("D:/PYE/file.txt") 180768</pre>

路径操作

`os.path.abspath(path)`

`os.path.normpath(path)`

`os.path.relpath(path)`

`os.path.dirname(path)`

`os.path.basename(path)`

`os.path.join(path)`

`os.path.exists(path)`

`os.path.isfile(path)`

`os.path.isdir(path)`

`os.path.getatime(path)`

`os.path.getmtime(path)`

`os.path.getctime(path)`

`os.path.getsize(path)`



os库之进程管理

进程管理

`os.system(command)`

- 执行程序或命令command
- 在Windows系统中，返回值为cmd的调用返回信息

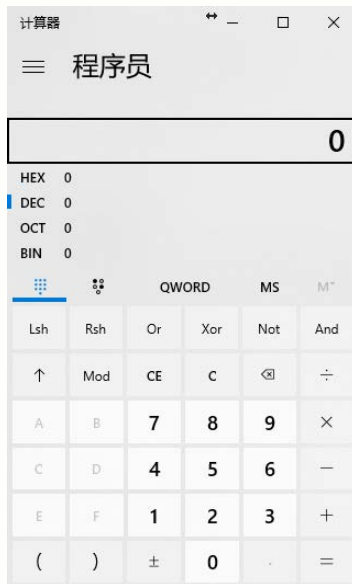
进程管理

```
import os
```

```
os.system("C:\\Windows\\System32\\calc.exe")
```

```
>>>
```

```
0
```



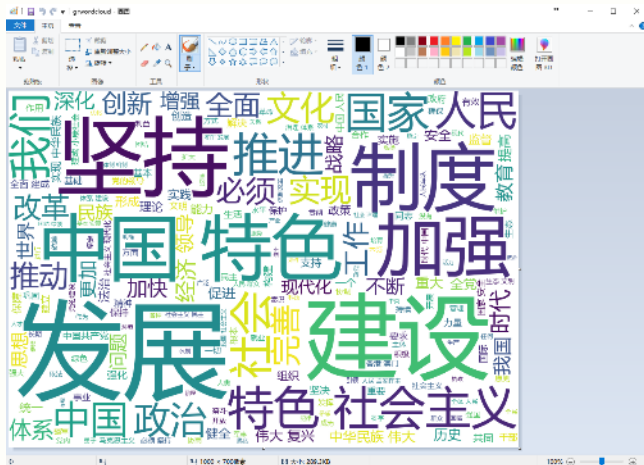
进程管理

```
import os
```

```
os.system("C:\\Windows\\System32\\mspaint.exe \\  
D:\\PYECourse\\grwordcloud.png")
```

```
>>>
```

```
0
```





os库之环境参数

环境参数

获取或改变系统环境信息

函数	描述
os.chdir(path)	修改当前程序操作的路径 <code>>>>os.chdir("D:")</code>
os.getcwd()	返回程序的当前路径 <code>>>>os.getcwd()</code> <code>'D:\\'</code>

环境参数

获取操作系统环境信息

函数	描述
os.getlogin()	获得当前系统登录用户名称 <pre>>>>os.getlogin() 'Tian Song'</pre>
os.cpu_count()	获得当前系统的CPU数量 <pre>>>>os.cpu_count() 8</pre>

环境参数

获取操作系统环境信息

函数	描述
os.urandom(n)	获得n个字节长度的随机字符串，通常用于加解密运算 <pre>>>>os.urandom(10)</pre> <pre>b'7\xbe\xfa!\xc1=\x01gL\xb3'</pre>

8.5 实例14: 第三方库自动安装脚本





"第三方库自动安装脚本"问题分析

问题分析

第三方库自动安装脚本

- 需求：批量安装第三方库需要人工干预，能否自动安装？
- 自动执行pip逐一根据安装需求安装

如何自动执行一个程序？例如：pip？

问题分析

第三方库自动安装脚本

库名	用途	pip安装指令
NumPy	N维数据表示和运算	pip install numpy
Matplotlib	二维数据可视化	pip install matplotlib
PIL	图像处理	pip install pillow
Scikit-Learn	机器学习和数据挖掘	pip install sklearn
Requests	HTTP协议访问及网络爬虫	pip install requests

问题分析

第三方库自动安装脚本

库名	用途	pip安装指令
Jieba	中文分词	pip install jieba
Beautiful Soup	HTML和XML解析器	pip install beautifulsoup4
Wheel	Python第三方库文件打包工具	pip install wheel
PyInstaller	打包Python源文件为可执行文件	pip install pyinstaller
Django	Python最流行的Web开发框架	pip install django

问题分析

第三方库自动安装脚本

库名	用途	pip安装指令
Flask	轻量级Web开发框架	pip install flask
WeRoBot	微信机器人开发框架	pip install werobot
SymPy	数学符号计算工具	pip install sympy
Pandas	高效数据分析和计算	pip install pandas
Networkx	复杂网络和图结构的建模和分析	pip install networkx

问题分析

第三方库自动安装脚本

库名	用途	pip安装指令
PyQt5	基于Qt的专业级GUI开发框架	pip install pyqt5
PyOpenGL	多平台OpenGL开发接口	pip install pyopengl
PyPDF2	PDF文件内容提取及处理	pip install pypdf2
docopt	Python命令行解析	pip install docopt
PyGame	简单小游戏开发框架	pip install pygame



"第三方库自动安装脚本"实例讲解

第三方库自动安装脚本

```
#BatchInstall.py
```

```
import os
```

```
libs = {"numpy","matplotlib","pillow","sklearn","requests",\  
        "jieba","beautifulsoup4","wheel","networkx","sympy",\  
        "pyinstaller","django","flask","werobot","pyqt5",\  
        "pandas","pyopengl","pypdf2","docopt","pygame"}
```

```
try:
```

```
    for lib in libs:
```

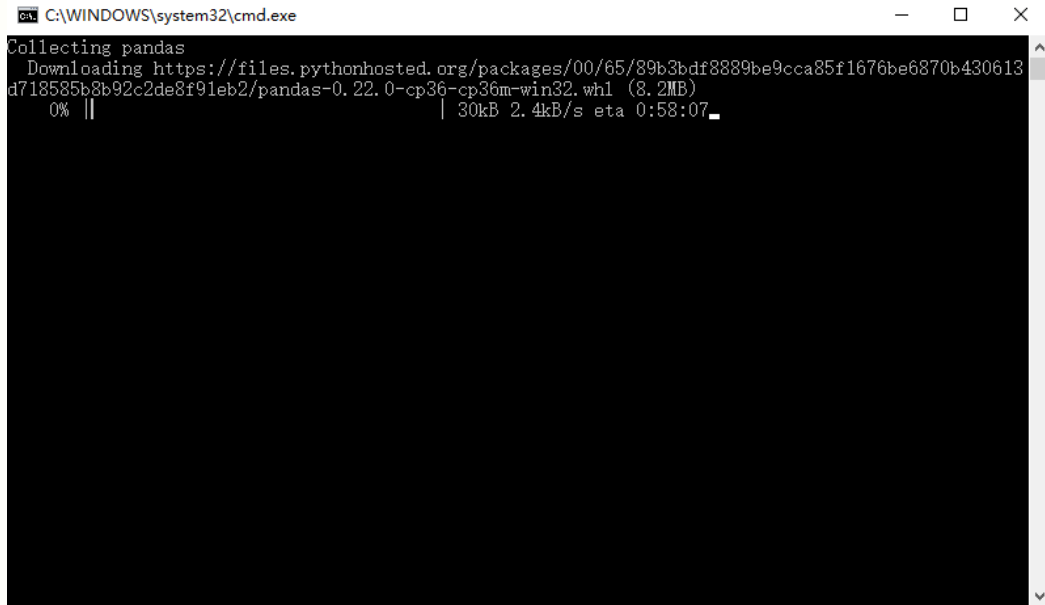
```
        os.system("pip install " + lib)
```

```
    print("Successful")
```

```
except:
```

```
    print("Failed Somehow")
```


第三方库自动安装脚本



```
C:\WINDOWS\system32\cmd.exe
Collecting pandas
  Downloading https://files.pythonhosted.org/packages/00/65/89b3bdf8889be9cca85f1676be6870b430613d718585b8b92c2de8f91eb2/pandas-0.22.0-cp36-cp36m-win32.whl (8.2MB)
    0% || | 30kB 2.4kB/s eta 0:58:07_
```



"第三方库自动安装脚本"举一反三

```
#BatchInstall.py
```

```
import os
```

```
libs = {"numpy","matplotlib","pillow","sklearn","requests",\  
        "jieba","beautifulsoup4","wheel","networkx","sympy",\  
        "pyinstaller","django","flask","werobot","pyqt5",\  
        "pandas","pyopengl","pypdf2","docopt","pygame"}
```

```
try:
```

```
    for lib in libs:
```

```
        os.system("pip install " + lib)
```

```
        print("Successful")
```

```
except:
```

```
    print("Failed Somehow")
```

举一反三

自动化脚本+

- 编写各类自动化运行程序的脚本，调用已有程序
- 扩展应用：安装更多第三方库，增加配置文件
- 扩展异常检测：捕获更多异常类型，程序更稳定友好

