Python

1. 安装

安装依赖包

#yum -y install gcc gcc-c++ zlib-devel openssl-devel readline-devel

libffi-devel sqlite-devel tcl-devel tk-devel

#tar -xf python-3.6.4.tar.gz

#cd pythob-3.6.4

#./configure --prefix=/usr/local

#make && make install

1. git版本控制软件

emal: 787560291@qq.com 账户名:songweihong132731 密码:boedt132731

#yum -y install git

# git config --global user.name "songweihong132731"

# git config --global user.emal "787560291@qq.com"

# git config --global core.editor vim

# git config --list

user.name=songweihong132731

user.emal=78756291@qq.com

core.editor=vim

# cat ~/.gitconfig

[user]

name = songweihong132731

emal = 787560291@qq.com

[core]

editor = vim

//以上为设置用户信息

创建仓库

#mkdir devops

#cd devops

#git init

添加文件到暂存区

#git add .

#git status -s

删除跟踪文件

#git ls-files //查看版本库中文件

#git rm hello.py

确认至仓库

# git commit -am "向hello.py添加新行"

# git commit -m "删除hello.py"

将git本地仓库和Github远程仓库关联

#git remote add origin git@github.com:boedt132731/1803.git

从远程仓库pull，获取远程仓库的文件到本地仓库

#git pull origin master

将本地仓库文件Push到远程仓库

# git push -u origin master

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

使用远程服务器-搭建gitlab服务器

添加密钥到github

#ssh-keygen

cat /root/.ssh/id\_rsa.pub

github网页右上角点击下拉菜单选择setting-->SSH and GPG keys.-->New SSH key-->Add SSH key

帮助文件：

(https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/)

测试连接

#ssh -T [Git@github.com](https://link.jianshu.com?t=mailto:Git@github.com" \t "/root/文档\\x/_blank)

测试成功结果：

You’ve successfully authenticated, but GitHub does not provide shell access 。

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

1. python语法结构

缩进相同的一组语句构成一个代码快

首行以关键字开始以冒号：结束

1. 解释器

#! /usr/bin/python

#! /usr/bin/env python #可以在默认环境外寻找解释器

1. 注释

# 单行注释

''' """python 中多行注释使用三个单引号(''')或三个双引号(""")。

1. 输入输出 文件I/O

input() raw\_input()

Print()

格式化输出%

文件处理

Open()打开 Close()关闭 Write() 写入read()读取//非默认路径打开方法

remove()

chdir() rmdir() mkdir()

1. 变量

变量赋值不需要类型声明，del语句删除对象引用

//type()函数查询变量所指的对象类型

//isinstance(变量，数据类型)判断变量的对象类型

五个标准的数据类型：

Numbers（数字）

int（有符号整型）

long（长整型[也可以代表八进制和十六进制]）

float（浮点型）

complex（复数）

String（字符串）

字符串的截取的语法格式如下：变量[头下标:尾下标]

单引号和双引号使用完全相同。使用三引号('''或""")可以指定一个多行字符串。

字符串可以用 + 运算符连接在一起，用 \* 运算符重复。

反斜杠可以用来转义，使用r可以让反斜杠不发生转义。

字符串操作符

比较操作府：字符大小按照ASCII码值大小进行比较

切片操作府：[] [:] [::]

成员关系操作符： in not in

格式化操作符

%c 转换成字符

%s 优先用str()函数进行字符串转换

%d / %i 转成有符号十进制数

%o 转成无符号八进制数

%e / %E 转换成科学计数法

%f / %F 转换成浮点数

\* 定义宽度或者小数点精度

- 左对齐

+ 在正数前面显示加号

<sp> 在正数前面显示空格

# 在八进制数前面显示0，在十六进制前面显示0x 或 0X

0 显示的数字前面填充0而不是默认的空格

format函数

使用位置参数 ‘my name is {} , age {}’.format(‘hoho’,18)

使用关键字参数 ‘my name is {name},age is {age}’.format({‘name’:’bob’,’age’:23})

填充与格式化 {:[填充字符][<对齐方式^>][宽度]}

使用索引 ‘name is {0[0]} age is {0[1]}’.format([‘bob’,23])

string.capitalize() 把字符串的第一个字符大写

string.center(width) 返回一个原来、字符串居中，并使用空格填充至width长

度的新字符串

string.count(str, beg=0, end=len(string)) 返回str在string里面出现的次数，如果beg

或者end指定则返回指定范围内str出现的次数

string.endswith(obj, beg=0, end=len(string)) 检查字符串是否以obj结束，如果beg

或者end指定则检查指定的范围内是否以obj结束，如果是，返回true，否则返回False

string.islower() 如果string中包含至少一个区分大小写的字符，并且所有字符都是

小写，则返回true，否则返回false

string.strip() 删除string字符串两端的空白

string.upper() 转换string中小写字母为大写

string.split(str=””, num=string.count(srt)) 以str为分隔符切片sting，如果num有指

定值，则仅分隔num个字符串

普通字符串和unicode字符串区别s

序列类型操作符

seq[ind] 获得下标为ind的元素

seq[ind1:ind2] 获得下标从ind1到ind2间的元素集合

seq \* expr 序列重复expr次

seq1+seq2 连接序列seq1和seq2

obj in seq 判断obj元素是否包含在seq中

obj not in seq 判断obj元素是否不包含在seq中

序列内建函数

list(iter) 把可迭代对象转换为列表

str(obj) 把obj对象转换成字符串

tuple(iter) 把一个可迭代对象转换成一个元组对象

len(seq) 返回seq的长度

max(iter，key=none) 返回iter中的最大值

enumerate 接受一个可迭代对象作为参数，返回一个enumerate对象

reversed(seq) 接受一个序列作为参数，返回一个以逆序访问的迭代器

sorted(iter) 接受一个可迭代对象作为参数，返回一个有序的列表

List（列表）

列表是写在方括号([])之间、用逗号分隔开的元素列表。

list可以被索引和切片、可以使用+操作符进行拼接、List中的元素是可以改变的。

//list = [ 'abcd', 786 , 2.23, 'runoob', 70.2 ]

list.append(obj) 向列表中添加一个对象obj

list.count(obj) 返回一个对象obj在列表中出现的次数

list.extend(seq) 把序列seq的内容添加到列表中

list.index(obj) 返回obj对象的下标

list.insert(index,obj) 在索引量为index的位置插入对象obj

list.reverse() 原地翻转列表

list.sort() 排序

Set（集合）

集合（set）是一个无序不重复元素的序列。

可以使用大括号 { } 或者 set() 函数创建集合。

set.add() 添加成员

set.update() 批量添加成员

set.remove() 移除成员

s.issubset(t) 如果s是t的子集，则返回True，否则返回False

s.issuperset(t) 如果t是s的超集，则返回True，否则返回False

s.union(t) 返回一个新集合，该集合是s和t的并集

s.intersection(t) 返回一个新集合，该集合是s和t的交集

s.difference(t) 返回一个新集合，该集合是s的成员，但不是t的成员

//parame = {value01,value02,...}

//set(value)

Tuple（元组）

元组是另一个数据类型，类似于List（列表）。元组用"()"标识。内部

元素用逗号隔开。但是元组不能二次赋值，相当于只读列表。

Dictionary（字典）

字典用"{ }"标识。字典由索引(key)和它对应的值value组成。

创建字典

adict={‘name’:’bob’,’age’:23}

adict=dict(([‘name’,’bob’],[‘age’,23]))

formkeys()创建具有相同值的默认字典

删除字典

del删除字典中的元素或整个字典

clear()方法可以清空字典

pop()方法可以“弹出”字典中的元素

len() 返回字典中元素的数目

hash() 本身不是为字典设计的，但可以判断某个对象是否可以作为字典的键

dict.copy() 返回字典(深复制)的一个副本

dict.get(key,default=None) 对字典dict中的键key，返回她对应的值value，如果字 典中不存在此键，则返回default的值

dict.setdefault(key,default=None) 如果字典中不存在key键，由dict[key]=

default为它附值

dict.items() 返回一个包含字典中(键，值)对元组的列表

dict.keys() 返回一个包含字典中键的列表

dict.values() 返回一个包含字典中所有值的列表

dict.update(dict2) 将字典dict2的键-值对添加到字典dict

5、 运算符

[算术运算符](http://www.runoob.com/python/python-operators.html" \l "ysf1)

\*\*幂 //取整除

[比较（关系）运算符](http://www.runoob.com/python/python-operators.html" \l "ysf2)

[赋值运算符](http://www.runoob.com/python/python-operators.html" \l "ysf3)

[逻辑运算符](http://www.runoob.com/python/python-operators.html" \l "ysf4)

[位运算符](http://www.runoob.com/python/python-operators.html" \l "ysf5)

把数字看作二进制来进行计算的

[成员运算符](http://www.runoob.com/python/python-operators.html" \l "ysf6)

[身份运算符](http://www.runoob.com/python/python-operators.html" \l "ysf7)

[运算符优先级](http://www.runoob.com/python/python-operators.html" \l "ysf8)

1. 条件语句

if 判断条件：

执行语句……

else：

执行语句……

if 判断条件1:

执行语句1……

elif 判断条件2:

执行语句2……

elif 判断条件3:

执行语句3……

else:

执行语句4……

//条件表达式 smaller=x if x<y else y

1. 循环语句

while 判断条件：

执行语句……

for 变量 in 判断条件:

执行语句......

1. 文件对象

打开文件open()和file()

file\_object=open(file\_name, access\_mode=’r’, buffering=-1)

文件访问模式：r、w、a、r+、w+、a、b（二进制模式打开）

文件输入

read() 直接读取字节到字符串中

data = fobj.read()

readline() 读取打开文件的一行

data = fobj.readline()

readlines() 读取所有（剩余的）行然后把它们作为一个字符串列表返回

data = fobj.readlines()

文件输出

write() 把含有文本数据或二进制数据块的字符串写入到文件中，不会自动添加行结束符

fobj.write(‘Hello world! \n’)

writelines 接受一个字符串列表作为参数，将它们写入文件，不会自动添加行结束符

fobj.writelines([‘Hello World!\n’, ’python programing\n’])

with语句简化代码，将打开文件的操作放在with语句中，代码块结束后，文件将自动关闭

with open(‘foo.py’) as f:

data = f.readlines()

seek(offset[, whence]): 移动文件指针到不同的位置

offset是相对于某个位置的偏移量；

whence的值，0表示文件开头，1表示当前位置，2表示文件的结尾

tell()：返回当前文件指针的位置

标准文件

标准输入，一般键盘，使用sys.stdin

标注输出，一般显示器缓冲输出，使用sys.stdout

标准错误，一般显示器非缓冲输出，使用sys.stderr

1. 函数

创建函数

def functionname( parameters ):

"函数\_文档字符串" //可选

function\_body\_suite

return [expression]

调用函数

functionname() //如果没有圆括号知识对函数的引用

参数

形式参数，函数定义时，括号内的参数

实际参数，调用函数时，括号内的参数

位置参数，使用sys模块的argv列表接收，程序名及参数都以位置参数的方式传递给程序

默认参数，声明了默认值的参数

1. 模块

模块的导入

Import 模块名

from 模块名 import 属性

//当解释器遇到 import 语句，如果模块在当前的搜索路径就会被导入。

// PYTHONPATH 通过sys.path查看

shutil模块

复制和移动

shutil.copyfileobj(fsrc,fdst[,length])

shutil.copyfile(src, dst, \*, follow\_symlinks=Ture)

shutil.copy(src, dst, \*, follow\_symlinks=Ture)

shutil.copy2(src, dst, \*, follow\_symlinks=Ture)

shutil.move(src, dst, \*, follow\_symlinks=copy2)

目录操作

shutil.copytree(src, dst, symlinks=False, ignore=none, copy\_function=copy2,

ignore\_dangling\_syslinks=false)

shutil.rmtree(path, ignore\_errors=F alse, onerror=none)

权限管理

shutil.copymode(src, dst, \*, follow\_symlinks=Ture)

shutil.copystat(src, dst, \*, follow\_symlinks=Ture)

shutil.chown(patt, user=none, group=none)

time模块

时间戳timestamp表示的是从1970年1月1日 00：00：00开始按秒计算的偏移量

UTC，世界标准时间

struct\_time属性

索引 属性 值

0 tm\_year 2000

1 tm\_mon 1-12

2 tm\_mday 1-31

3 tm\_hour 0-23

4 tm\_min 0-59

5 tm\_sec 0-61

6 tm\_wday 0-6(0表示周一)

7 tm\_yday(一年中的第几天) 1-366

8 tm\_isdst(是否为dst时间) 默认为-1

time.localtime([secs]) 将一个时间戳转换成当前时区的struct\_time

time.gmtime([secs]) 将一个时间戳转换为UTC时区的strct\_time

time.time() 返回当前时间的时间戳

time.mktime(t) 将一个struct\_time转化为时间戳

time.sleep(secs) 线程推迟指定的时间运行，单位为秒

time.asctime([t]) 把一个表示时间的元组或者struct\_time表示为以下形式：

Sun Jun 20 23：21：05 1996

time.ctime([secs]) 把一个时间戳(秒计算的浮点数)转化为time.asctime()的形式

time.strftime(format[,t]) 把一个代表时间的元组或者struct\_time转化为格式化的时间

字符串

time.strptime(string[,format]) 把一个格式化时间字符串转化为struct\_time

时间样式

格式 含义

%a 本地简化星期名称

%A 本地完整星期名称

%b 本地简化月份名称

%B 本地完整月份名词

%c 本地相应的日期和时间

%d 一个月中的第几天（01-31）

%H 一天中的第几个小时（24小时制，00-23）

%I 第几个小时（12小时制，01-12）

%j 一年中的第几天（001-366）

%Z 时区的名字

%m 月份（01-12）

%M 分钟数(00-59)

%p 本地am或者pm的相应符

%S 秒（01-61）

%U 一年中的星期数(00-53，星期日是一个星期的恺撒)

%w 一个星期中的第几天(0-6, 0是星期天)

%x 本地相应日期

%X 本地相应时间

%y 去掉世纪的年份（00-99）

%Y 完整的年份

datetime模块

datetime.today() 返回一个表示当前本地时间的datetime对象

datetime.now([tz]) 返回一个表示当前本地时间的datetime对象，如果提供了参数tz，

则获取tz参数所指时区的本地时间

datetime.strptime(date\_string,format) 将格式字符串转换为datetime对象

datetime.ctime(datetime对象) 返回时间格式字符串

datetime.strftime(format) 返回指定格式字符串

datetime.timedelta 在日期上作天，小时，分钟，秒，毫秒，微秒的时间计算

OS模块

对文件系统的访问大多通过python的os模块实现

该模块是python访问操作系统功能的主要接口

symlink() 创建符号链接

listdir() 列出指定目录的文件

getcwd() 返回当前工作目录

mkdir() 创建目录

chmod() 改变权限模式

getatime() 返回最近访问时间

chdir() 改变工作目录

pickle模块

把数据写入文件时,常规的文件方法只能把字符串对象写入。其他数据需先转换成字符串再

写入文件。pickle模块可以在一个文件中储存任何python对象，之后又可以把它完整无

缺地取出来

分别调用dump()和load()可以存储、写入

pickle.dump()

pickle.load()

1. 异常处理

try/except语句用来检测try语句块中的错误，从而让except语句捕获异常信息并处理。

try:<语句> #运行别的代码

except <名字>：

<语句> #如果在try部份引发了'name'异常

except <名字>，<数据>:

<语句> #如果引发了'name'异常，获得附加的数据

else:

<语句> #如果没有异常发生

1. 面向对象编程

**类(Class):** 用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例。

class ClassName:

'类的帮助信息' #类文档字符串

class\_suite #类体

导入类

Form 模块名 import 函数名

\_\_init\_\_初始化构造方法。在调用了一个实例的方法时，该方法的self参数会自动绑定到实例上（称为绑定方法）

**类变量：**类变量在整个实例化的对象中是公用的。类变量定义在类中且在函数体之外。类变量通常不作为实例变量使用。

**数据成员：**类变量或者实例变量, 用于处理类及其实例对象的相关的数据。

方法重写：如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程叫方法的覆盖（override），也称为方法的重写。

**实例变量：**定义在方法中的变量，只作用于当前实例的类。

**继承：**即一个派生类（derived class）继承基类（base class）的字段和方法。继承也允许把一个派生类的对象作为一个基类对象对待。例如，有这样一个设计：一个Dog类型的对象派生自Animal类，这是模拟"是一个（is-a）"关系（例图，Dog是一个Animal）。

**实例化：**创建一个类的实例，类的具体对象。

**方法：**类中定义的函数。

**对象：**通过类定义的数据结构实例。对象包括两个数据成员（类变量和实例变量）和方法。