python

交互式语言 可以逐句执行

表达式

a+b 加法
a-b 减法
a*b 乘法
a/b 除法
a//b 整除
a**b 乘方运算
123L 长整型
abs(a) 绝对值
int(a) 转换为整数
long(a) 转换为长整
float(a) 转化为浮点型
round(a,n) 四舍五入保留 n 位 默认零位取整

字符串

字符串是不可变的
\为转义符号
“ ‘ 通用
a+b 拼接字符串 两个都是字符串
str(a)将 a 转化为字符串
repr(a)将 a 转化为字符串带引号类型类型标识
r”asd” 原始字符串 转义符无用 先有字符串再有原始 所以转义会对末尾”有用
u”asd” unicode 字符串 python3 所有字符串均为 unicode 字符串

格式化

字符串%元组 的方式进行格式化
前面替换的用%s 字符串 %3.3f 浮点
字符串中带%的话用%%
若用字典格式化需在%和 s 中间加入键的名字

a.find(“asd”) 查找子串 返回第一个匹配项最左端索引 未找到返回-1
“/”.join(“a”,”b”,”c”) 使用/连接 abc
a.lower() 返回小写版本
a.replace(“asd”,”zxc”) 将所有 a 中的 asd 替换为 zxc 返回新的串
a.split(“+”) 将 a 按+切分 加号舍去 返回序列
a.strip(可选字符串) 返回两边的空格 可选择字符串里的字符作为清空字母 返回新的串
a.translate() 也用于替换 有替换表

语法

语句块：相同的空格或制表符后移作为子句 类似于大括号
使用 pass 可以什么都不执行

使用 del 语句进行删除变量 只是删除变量名 内容不删除
赋值语句:
a=3
打包: 可以使用 a=1,2,3 这种方式进行打包 变为元组
解包: 可是使用 a,b,c=(1,2,3)进行解包 abc 分别为 1,2,3
如左边元素不够 可以在一个变量左侧使用*将剩下值都放进该变量
可以使用 x+=1 进行加法

条件语句:
if 条件:
语句、
条件为否时 是 False None 0 “” () [] {}

比较运算
x==y 等于
x<y 小于
x>y 大于
x.>=y 大于等于
x<=y 小于等于
x!=y 不等于
x is y 是同一个对象
x is not y 不是同一个对象
x in y x 是 y 的成员 y 是容器
x not in y x 不是 y 的成员 y 是容器
比较运算可以连接 1<x<3
条件运算符 a if b else c 如果 b 则 a 否则 c

循环语句:
while 语句
while 条件:
语句
for 语句
a=[1,2,3]
for x in a:
语句

for 语句用于迭代序列
使用 range(起始，末尾)不包含末尾 产生一个序列
python3 不包含 xrange
循环字典元素 for key in a : pass
key 为键
可以使用 for key,value in a.items(): 进行循环
使用 zip(a,b)进行两个序列的压缩 相应每对元素组合为元组 任意一个序列用完就停止
使用 enumerate (a) 进行遍历序列 返回索引和值元组的列表

后面可以加 else 仅在并非 break 跳出的情况下执行

列表推导式:

[x*2 for x in a] 得到一个列表 列表从 a 列表的每个元素进行运算得到的

[x*y for x in a for y in b] 可以使用两个循环 x 嵌套 y

[x*2 for x in a if x>m] 可以添加条件

断言语句:

assert 1<x<3 ,"提醒信息"

若不符合上边的条件则异常 AssertionError

#作为注释符号 单行

换行可以用\将换行符转义 达成两行一句的效果

列表方法 对象.方法的方式

执行字符串代码:

使用 exec(a) 将 a 字符串作为代码进行执行

使用 eval(a) 将 a 表达式计算出值并返回

函数:

def 函数名(参数列表) :

处理

文档化: 可在 def 下面加入字符串注释使用""" """进行注释 并记录在__doc__属性中

返回值: 使用 return 进行返回 所有函数若不 return 或 return 空则返回 None

参数:

实参传递进到实参 更改形参不会改变实参值 除非传入列表等带有子元素的变量

使用关键值参数: 传递参数时可以指定参数名字, 不必完全按照位置来进行传递,

尽量不要混用位置参数和关键值参数

使用默认值 在形参后写好默认值

收集参数

在形参前加*

使用形参前*进行位置参数的收集 只能放到最后将剩下的所有参数收集

不能赋默认值

不能使用关键值参数赋值

收集参数会返回元组 若无多余参数则返回空元组

使用形参前**进行关键值参数的收集 只能放到最后将剩下的所有关键值参数收集

关键值收集参数会返回字典 若无多余参数则返回空字典

分配参数

收集参数的逆过程

使用实参前*进行元组或列表的分配 形参不加星号

使用实参前**进行字典的分配 形参不加星号

lambda 表达式：用于匿名的短小的函数
x=lambda 参数列表: 表达式
调用 x(参数列表)

作用域:
每个函数会创建一个自己的内部作用域
内部创建变量不会影响外部变量
内部可以引用全局变量

globals()返回全局变量字典
locals()返回局部变量字典
vars()返回当前作用域变量字典

若要赋值全局变量 则可以使用 global x 进行声明
若要赋值非局部非全局变量上层作用域 则可以使用 nonlocal x 进行声明（跟调用无关）

类:
多态是 python 式编程的核心
类定义:
class 类名:
方法 1（self,列表）
方法 2（self,列表）
方法 3（self,列表）
类本身是执行代码块 可以写入执行代码 变量等 定义时执行
每个类都有自己的名称空间

构造函数
使用__init__()来进行定义 对象构造时自动调用
使用 super(父类名,self)的方式调用父类对象
属性
self 指代的对象本身（不是类）
方法需要 self 参数
可以使用 class.method(对象）的方法非绑定的调用方法
property 函数将一个虚拟属性（由其他实际属性组合）变为可调用的
可使用@property 将方法变为属性直接调用 可通过此方法设置为只读属性
静态方法 函数上方写@staticmethod 不能访问实例变量以及类变量
类方法 函数上方写@classmethod 不能访问实例变量 可以访问类变量

使用 hasattr(对象,方法或特性名字符串）判断是否有该方法或特性
使用 getattr(对象，方法或特性名字符串，”“默认值”）获得方法或特性
使用 setattr(对象，方法或特性名字符串，”“设置的值”）设置方法或特性

基本序列映射规则:

查询长度：__len__(self):
访问元素：__getitem__(self):
设置元素：__setitem__(self):
删除元素：__delitem__(self):

调用拦截
__getattribute__(self,name):当 name 被访问时调用
__getattr__(self,name):当 name 被访问且对象没有相应特性时调用
__setattr__(self,name,value):当试图给 name 赋值时调用
__delattr__(self,name):删除特性时调用

迭代器
实现了__iter__(self):
返回的内容可以进行迭代
若本类实现了__next__() 则可 return self
实现了__next__(self)则为迭代器
每次调用返回一次结果
用完时引发 StopIteration 异常
自己定义的手动 raise StopIteration
可使用 iter(a) 将 a 转化为迭代器 a 为序列
可使用 list(a) 将 a 从迭代器转化为列表
生成器
在一个函数中使用 yield 该函数将成为一个迭代器
会自动保存临时状态 可继续 for 循环语句
def a():
for i in data:
yield i

使用__dict__查看所有方法和特性
使用__str__进行 print 打印输出内容
使用__repr__进行 print 打印输出内容
使用__call__将对象为可调用的

继承：
定义超类：class 类名（父类名）：
可以子类写父类的函数达到重写的目的
可以直接调用父类的函数
使用 issubclass（子类名，父类名）检查是否是子类
使用 isinstance（对象名，类名）检查是否是实例
类的__bases__属性记录了所有父类
对象的__class__属性记录了对象的类
多继承：
若两个类函数名冲突 先写的会覆盖后写的

私有化:

若让方法私有化 函数名前加__ 解释器将函数改名字达到不能访问的目的

使用_则是子类可以访问

可以使用_ 导入时不会被 from 模块名 import *的语句导入做不到真正的私有化

__metaclass__=type 显式定义新类 python3.0 以后无新旧之分

序列

属于一种容器

通用操作:

索引: 通过 [i] 的方式进行查找 i 从 0 开始 通过-1 -2 从后往前数

分片: 访问一个范围的元素 通过[i:j]的方式, 返回序列 i 开始到 j 之前, 不包含 j

若分片包含结尾元素可使用结尾的下一个坐标或者直接置空, 也可以开始置空回从头开始的序列, 若都置空 则返回所有元素

可加入步长 通过加入第三个索引 形成[i:j:k] k 为步长 从 i 开始每 k 个元素到 j 之前

拼接: 通过 a+b 达成拼接的目的 相同类型 返回新的对象

乘法: 通过对列表乘法可以得到原先列表重复 n 次的列表 可以通过内建类型 None 来占位

是否存在(成员资格): 使用 a in b 返回布尔值判断 a 是否在 b 中

长度: len(a)

最大值: max(a)

最小值: min(a)

列表:

使用 [ ]

list(a)函数将 a 变为列表、

赋值: a[0]=1

删除: del a[0]可直接删除该元素 后面元素索引往前移动

分片赋值: a[1:4]=[0,1,2,3,4,5] 可以不等长 多出来的部分插入进去 后面元素索引相应移动 可以 a[1:1]=[1,2,3]来达到的插入的目的 也可用 a[1:2]=[]来达到删除的目的

a.append(x) 末尾加入对象

a.count(x) 对某个元素进行统计计数

a.extend(x) 末尾加入序列 返回修改过的 a

a.index(x) 返回第一个 x 匹配项的索引 若没有找到则抛出异常

a.insert(loc,x) 在 loc 位置后 插入 x

a.pop(loc) 弹出 loc 位置 (可不加位置默认-1) 的元素并返回

a.remove(x) 删除 x 第一个匹配的元素 若没有则抛出异常

a.reverse() 将所有元素反向存放

a.sort() 排序 不返回值

元组:

除了不能更改 其他的列表相似

使用 ( ) 一个元素要使用“, ”逗号

tuple(a)将 a 转换为元组、
元组推导式：类似于列表推导式

字典

不是序列 但是类似
使用{} 直接定义
dict(a)函数将 a 变为字典
通过元组列表进行创建 a=[('a',1),('b',2),('c',3)] 结果为{a:1,b:2,c:3}
使用关键字创建 dict(a=1,b=2,c=3)
键可以是任意不变类型 基本类型 包括元组
长度：len(a) 返回字典元素的个数
索引：a["m"] 返回键为 m 的值 可进行赋值
添加：a["n"]=5 直接对新键赋值即可添加
删除：del a["m"]
查看存在："m" in a 查看键为 m 的元素是否在 a 中存在
a.clear() 清除列表所有项
a.copy() 浅复制 复制父对象 不复制子对象
a.deepcopy() 深复制 from copy import deepcopy 复制父对象 子对象
{}.fromkeys(['a','b','c'], 可选初始值) 用给定键创建新的字典 值均为 None 原字典不改变
可以直接将{}换为 dict
a.get('m',可选返回值) 更为自由的获取值的方法
若找不到键值则返回 None 或可选返回值 不会抛出异常
a.has_key("m") 检查是否特定键 python3 没有该函数
a.items() 该方法将每一个项变为键值元组以列表的方式返回（需要用 list()将 dict_items 类型
变为 list）没有顺序
a.iteritems() 返回迭代器 效果同 items
a.keys() 返回键的列表
a.iterkeys() 返回键的迭代器
a.pop("m") 弹出返回并删除相应键的值
a.popitem() 随机弹出项 逐个处理起来很方便 返回元组第一个元素 key 和第二个 value
a.setdefault("m",1) 若 m 存在则返回 若不存在则设为·1
a.update(b) 用 b 更新 a 有相同键名的项

字典推导式：类似于列表推导式

异常

抛出异常
程序自动抛出异常
使用 raise 手动抛出异常
raise 异常类("输出信息")
在 except 中直接 raise 不加参数重新抛出异常
自定义异常
需要继承 Exception 类 class myexception(Exception):pass
捕捉异常

try:
    语句
except 异常类 1:
    处理
except 异常类 2:
    处理
except (异常类 3,异常类 4) as e :
    处理
else:
    处理
finally:
    处理
可以直接 except : 捕捉所有异常
可以使用 else 进行若没有错误时的处理
finally 一定会被执行
若访问异常对象 可在异常类后面加入 as 变量名 变量名作为对象引用

模块

引用:
    import A 导入 A 模块
        将创建新的命名空间
    from A import B 从 A 中导入 B 函数
        不创建新的命名空间 而直接导入到当前空间
    from A import * 从 A 导入全部
    import A as a 使用别名
    默认只引用一次 可以 reload 多次引用 python3.0 已经不支持
    使用时则用 模块名.函数 的方式
查找路径:
    添加查找路径 sys.path.append("地址") 绝对地址 本次运行有效
    查看查找路径 sys.path
    sys.path 由 PYTHONPATH 和默认路径和当前路径加在一起
    大部分模块存在 python3.x\\lib\\site-packages 中
    添加地址:
    添加 sys.path
    在 site-packages 中添加 pth 文件 文件名任意 并写入想要的目录
    在用户变量中用 PYTHONPATH 添加地址

显示当前的调用方式: __name__变量 若是"__main__"则是以主程序调用 若是模块名则是以
    模块调用
设置接口: 使用__all__设置接口 将被*引用 代表了所有对外使用的函数及变量

查看所有变量：使用 dir(模块名) 查看所有模块的变量和函数
查看模块位置：可以查看__file__属性查看文件的位置

pyc 文件是模块的编译后文件

包：
使用__init__.py 文件作为包的说明文件
可以将其作为包的内容 直接引用 包 a b 在__init__中 使用 a.b 的方式引用
引用
import 包名.模块名
from 包名 import 模块名
from 包名.模块名 import 函数名

文件

打开文件 f=open(“文件名”,”mode”,buffering,encoding)
可设置是否进行缓冲 0 为不缓存
可设置 encoding 进行编码
关闭文件 f.close()
写内容 f.write(内容)
读内容 f.read(字节数)
读一行 f.readline() 遇到换行符读取换行符并停止
读多行 f.readlines() 返回每行的 列表
写多行 f.writelines([字符串列表])
使用 with open(名字) as f :的方式（上下文管理器）进行打开
移动指针 f.seek(offset,whence)offset 向后的位置数 whence 从哪里开始算
刷新缓存 f.flush()

网络

日志

导入模块 import logging
设置参数 logging.basicConfig(level=logging.INFO,filename="log1.log")
记录 logging.info("start")

cython

cython 语法是 python 的超集 包括 python 和 c 的变量定义
编译好后可以像一般模块一样调用
编译的语法：
文件 setup.py
from distutils.core import setup
from Cython.Build import cythonize
setup(ext_modules=cythonize("被编译文件")) 被编译文件名为 xxx.pyx
编译 python setup.py build_ext --inplace 生成 xxx.so
可以在调用文件中使用 import pyximport 和 pyximport.install()省去编译步骤
cython 语法：

可以直接使用 python
可将变量之前使用 cdef 类型 名称 的方式声明变量 使用方式与 python 一样

cython 调用 c 语言：
自带 c 语言库 libc libcpp 使用如 from libc.stdlib cimport atoi
使用 cdef extern from “xxx.h”:
int a()
cppclass b:
b()
char* c()
的方式调用自己写的 c 文件

函数大全

获取输入 input(“message”)可以输出提示信息
获取纯输入 raw_input(“message”) 不进行转义等操作 可以输出提示信息 pyhton3 之前
进行输出 print(“message”)
可以使用 print (a , b) 进行不同类型变量输出 a b 输出之间会有空格
更智能的输出 pprint(“message”)
import pprint
pprint.pprint(“message”)
打印不下自动换行

获取反向的序列：reversed(x) 返回副本
获取序列的排序：sorted(x) 返回副本

获得函数帮助和信息注释：help(x) x 为函数名
获得当前作用域变量的内建字典：vars()
获得全局变量的内建字典：globals()
获得局部变量的内建字典：locals()

查看模块所有函数和变量：dir(模块名)

检查是否是子类：ssubclass（子类名，父类名）
检查是否是实例：isinstance（对象名，类名）

判断是否有该方法或特性：hasattr(对象,方法或特性名字符串)
获得方法或特性：getattr(对象，方法或特性名字符串，”“默认值”)
设置方法或特性：setattr(对象，方法或特性名字符串，”“设置的值”)

sys
访问与解释器相关联的变量和函数
sys.argv 变量 传递参数
sys.exit(参数信息) 函数 退出时返回信息

sys.path 列表 记录模块查找路径
sys.platform 一个字符串记录操作系统名字
sys.stdin sys.stdout 文件输入输出

os

调用操作系统服务
os.environ 系统环境变量
os.system(command) 子 shell 里执行命令 command 为指令字符串
os.sep 当前系统中的分隔符
os.name 操作系统名称
目录操作:
os.getcwd() 当前工作路径
os.listdir(path) 返回目录下的文件以及目录信息
os.mkdir(path) 创建目录 若上一级目录不存在则抛异常 若存在也抛异常
os.makedirs(path) 创建目录 若上一级目录不存在则创建上一级目录
os.path.abspath(path) 获取文件的绝对路径
os.path.join(path1,path2) 将两个目录拼接
os.path.exists(path) 该路径是否存在
os.rmdir() 删除目录 目录要为空
os.walk(path,) 返回（当前路径，子目录列表，文件列表）的生成器
for a,b,c in os.walk(path):
os.listdir(path) 返回当前路径的所有文件夹和文件
os.path.isdir(path) 判断是否是文件夹
os.path.isfile(path) 判断是否是文件
os.access(path,accessmode) 对某文件是否有具体权限
os.shmod(path,mode) 修改权限
os.remove(path)删除文件
os.rename(srcname,dstname) 重命名
os.stat(path) 获取文件信息 返回包含信息的对象

shutil

高级文件操作
shutil.move(src,dst)移动文件
shutil.copy(src,dst)复制文件 dst 可以是文件夹
shutil.copytree(src,dst)复制文件夹
shutil.rmtree(path)删除文件夹

time

时间相关的函数
localtime 当前时间数组 可根据索引访问
sleep(秒数) 休眠
time() 当前时间 新纪元开始后的秒数

urllib

用于读取来自网络上的数据的标准库

json

用于使用 JSON 序列 反序列化

re

正则表达式
re.compile(模式字符串) 编译字符串
re.findall(pattern,str) 列出所有匹配项
re.sub(被替换模式,替换模式,字符串)
语法:
"."通配符
[a-bd-f]使用中括号规定具体匹配的符号和范围 使用^进行匹配括号外的
p(abc|bcd) ()为子模式 使用|可选择模式内的串
(abc)?bcs 子模式后使用? 代表可出现 可不出现
(asd)+ (asd)* (asd){a,b}+代表重复一次或者多次 *代表重复 0 次或多次 {a,b}代表 a,b 次

math
向上取整 ceil(a)
向下取整 floor(a)
四舍五入 round(a)

cmath
求平方根 sqart(a)

decimal
控制精度 有效位数 四舍五入的十进制运算

logging
提供了灵活的记录调试信息的日志功能

tkinter
gui 编程的标准库

random
随机数
random() 产生 0 到 1 的随机数
uniform(a,b) 产生 a 到 b 的随机数
choice(序列) 从序列返回随意元素
sample(seq,n) 从序列中选择 n 个随机独立的元素

shelve
快速的文件存储 通过字典的方式
f=shelve.open("文件名") 打开文件
s['字典标签']=值
s.close()关闭文件

二分查找 bisect

pillow:
from PIL import image
打开图片 im=Image.open("名字")
保存图片 im.save("名字", 可选格式)
显示图片 im.show()
图片大小 im.size

图片模式 im.mode
选择一个框 box=im.crop((int,int,int,int))元组
粘贴一个框 im.paste(box, (int,int,int,int))元组
分离颜色通道 r,g,b=im.split()
合并颜色通道 Image.merge("RGB",(r,g,b))
放大缩小图像 im.resize((int,int))
旋转图像 im.ratate(int)角度
im.transpose(Image.xxxxx)直接处理
模式转变 im.convert("模式")
过滤器
    from PIL import ImageFilter
    out=im.filter(ImageFilter.DETAIL)
点操作 im.point()
效果增强
    from PIL import ImageEnhance
    enh=ImageEnhance.Contrast(im)创建实例
    enh.enhance(1.3).show("30% more constrast")
输出图片
    Postscript
        from PIL import PSDraw
        ps=PSDraw.PSDraw()
        ps.begin_document("")
        ps.image((int,int,int,int),im,分辨率)
        ps.rectangle(box)

        # draw title
        ps.setfont("HelveticaNarrow-Bold", 36)
        ps.text((3*72, 4*72), title)
        ps.end_document()
从字符串读取图片
    import StringIO
    im=Image.open(StringIO(buffer))