python3 廖雪峰

# 1.数据类型

## 1.1.基础类型

* 整数
  + 普通整数：1, -1, 2, -200等；
  + 十六进制：0xff00, 0x1234,等；
* 浮点数
  + 普通浮点数：1.23，2.35，等；
  + 很大的浮点数：例如1.23\*109就是1.23e9；
  + 很小的浮点数：例如0.00012就是1.2e-4；
* 字符串：以一对单引号或一对双引号括起来的文本
  + 普通字符串：”abc”，’deg’等；
  + 单引号或双引号也在字符串内的表示：”abc’def’gdh”，’wed”eogi”df’等；
  + 转义字符：
    - “abd\”def”，这里中间的双引号因为加了转义，所以作为一个字符对待，而不是字符串的结尾对待，单引号同理；
    - \n表示换行；\t表示制表符；\\表示”\”这个字符；
    - 使用r””或者r‘’可以禁止字符串中所有字符被转义，例如r”\t”输出的就是\t这个字符串，而不是一个制表符；
* 布尔值
  + True，False；
  + 逻辑与and，逻辑或or，逻辑非not；
* 空值
  + None表示空值，不能理解为0，因为0有意义，而None没有任何意义，只是python自己定义的一个特殊的空值；
* 常量
  + 定义常量：PI=3.1415926,但是PI其实还是一个变量，可以任意改变，只是我们这么约定它是个常量；
  + 整数除法：
    - 10/3=3.3333333…，默认的除法是四舍五入的一个浮点数作为结果；
    - 10//3=3，地板除，可以得到整数除法；
    - 10%3=1，求余；

## 1.2.高级类型

### 1.2.1.list-列表

列表，有序，可以随时增加和删除元素，由一对方括号作为开始和结束的标记：a=[1,2,3], b=[“1”, “2”, ‘3’]；

支持的基本操作罗列如下：

* 获取列表长度，使用len()函数，例如len(a)；空列表的长度为0；
* 可以使用下标直接访问列表中的元素，例如a[0]表示读取a中的第一个元素；
* 支持从后面开始计数的下标，例如a[-1]表示最后一个元素；
* 如果索引下标超出了支持的范围，python会上报IndexError的错误，例如a[7]；
* 追加元素到列表尾部：a.append(4);
* 插入元素到某一个指定位置:a.insert(1, 6)，a的值就会变成[1, 6, 2, 3];
* 删除列表末尾的元素：a.pop()
* 删除特定下标的元素：a.pop(3);
* 更改某个元素的值：a[1] = 4;
* **list里的数据类型可以不同**，例如：c=[1, “a”, [‘d’, 2]]；

### 1.2.2.tuple-元组

tuple同样是一种有序列表，与list最大的区别是：**tuple一旦初始化，就不能再修改！**

tuple可以看作一个const的list，除了不能增删元素、修改元素的值，其他属性无差异；

tuple的不能修改的特性，使得我们可以更安全的操作数据；

需要注意，如果是空元组，直接使用a=()就可以声明；但**如果是一个元素的元组，需要用a=(1,)声明，逗号一定不能少**，否则会被误认为是一个基础数据类型；

### 1.2.3.dict-字典

dictionary，字典，C++中为std::map，使用键值对(key-value)存储，具有极快的查找速度；其定义，采用了一对花括号；

例如 d={“a”:5, “b”:6}，那么我们要查找”a”的值，就可以直接使用d[“a”]来实现；

可以动态增加dict中的元素，例如：d[“d”] = 7；

也可以修改通过如上的方式修改value的值，例如：d[‘d’]=8；

如果要查找不存在的key，dict会报错，要防止这种错误，有两种方式：

1. 通过in判断key是否存在于dict中，例如：”e” in d: 将返回一个布尔值，通过返回值判断key是否存在即可；
2. 通过dict自己提供的get方法，如果key不存在就返回None，例如d.get(“e”)，如果不存在就返回了None；

需要注意，第二种方法中，get可以指定返回值，例如不希望返回None，而是返回-1，可以：d.get(“e”, -1)

另外，key必须是不可变的，list是一个可变的类型，因此不可以作为key使用！

# 2.函数

## 2.1.运算符

### 2.1.1.条件判断

if a >= 3:

…

else

…

如上为最简单的一种条件判断；

还支持elif来做多级条件判断；

if这一行最后一定要用“：“结尾！

### 2.1.2.循环

* for循环：
  + name=[“a”, “b”]; for name in names: print(name);这种使用list或tuple中的元素，进行循环；
  + for I in range(1, 5): print(i);这种使用了range()函数来进行遍历的方式更简洁，要注意，range(a,b)定义的，是[a, b)这个区间范围；
* while循环：只要条件满足，一直执行，例如：while a > 0 : ….

支持使用break退出循环体；

支持使用continue跳出本次循环；

## 2.2.函数

### 2.2.1.定义函数

定义一个函数，需要依次写出：def关键字，函数名，函数参数(以括号括起)，冒号，函数体；

例如：

def func(x):

print(x)

支持通过return返回返回值；且执行到return时函数将返回，后续代码不予执行；

### 2.2.2.桩函数

可以使用pass关键字定义一个桩函数，也就是空函数，实现编译通过的目的，并不影响其他模块的编写和测试，例如：

def func(x):

pass

### 2.2.3.传入参数校验

对于参数个数的校验，python可以自动完成，例如：

def func(x):

pass

func(1, 2)这种调用方式，将会抛出TypeError的错误。

但是对于参数类型的校验，python无法自动校验。我们可以在函数体内自动进行：

def func(x):

if not isinstance(x, (int, float)):

raise TypeError(“bad operator type!”)

pass

这时候就可以校验传入参数的类型，如果不是int或者float，将抛出TypeError的异常；

isinstance函数，是python内置的数据类型检查函数；

### 2.2.4.返回多个返回值

def func(x):

return x, y, z

这种形式，可以返回形式上的3个返回值，但其实这里返回的，是一个元组！也就是说，我们可以用两种方式接收返回值：

1. x, y, z = func(x);
2. r = func(x)；

第二种方式里，print(r)，输出的一定是3个元素的元组；

可以这样写，是因为python支持tuple返回括号，仅此而已。

### 2.2.5.默认参数值

可以使用默认参数值的方式，简化程序调用，例如：

def func(x, y=2):

pass

这样，调用时如果使用func(1)，那么第二个参数就是用默认的2；如果使用func(1,3)，那么第二个参数就是用传入的3；

与C++相同，默认参数必须在函数参数的尾部；也可以支持多个，但必须在尾部，例如：

def func(a, b, c=”a”, d = 6):

pass

但是，默认参数，必须指向不可变的对象！不要指向列表等可变的对象!

可以通过指定名字，使用默认参数，例如：func(1, 2, d=7)这种调用方式，第三个参数就默认使用了c=”a”；

### 2.2.6.递归函数

def fact(n):

if n == 1:

return 1

return n \* fact(n - 1)

如上是简单的递归函数。

递归调用，最重要的是要防止栈溢出！

# 3.高级特性

## 3.1.切片

可以通过切片操作，取得某个对象中的特定位置的元素:

* list：L[1:3]
* tuple：t[:-2]
* 字符串：s=”shgeghoahgprhg”; s[3:-1]

需要明确的要点有：

1. 切片[a:b]的取值范围，是[a,b),也就是说并不包含b在内；
2. 切片[:b]省略了开始的位置，就默认是0，也就是从第一个元素开始；
3. 切片支持从后向前的计数方式，也就是支持负数传入；

## 3.2.列表生成式

列表生成式，list comprehensions，是python内置的非常简单却强大的可以用来**创建list**的生成式。

例如，我们想要创建一个[1\*1, 2\*2, 3\*3, …, 11\*11]的list，首先可以用函数使用for循环创建，但更好的方式是使用列表生成式：

[x \* x for x in range(1, 12)]

这样就直接生成了我们需要的list了！

其格式是：[要生成的元素 for循环];

我们可以只要偶数的平方：[x \* x for x in range(1, 12) if x % 2 == 0]；

也可以使用多个参数：

d = {a : 1, b : 2, c: 3}

[k + “=” + v for k, v in d.items()]

# 4.函数式编程

## 4.1.函数指针

python中，支持将函数名称赋值给一个变量，等同于C++中的函数指针，使用该变量可以直接执行函数功能。

例如，abs函数，用来求绝对值，可以如下赋值：

func = abs

也可以将函数名称作为参数，传递到其他函数使用，例如：

def add(a, b, func):

return func(a) + func(b)

这里就将func作为一个参数，传递进来。

## 4.2.map函数

map函数接收两个参数:第一个参数是一个函数指针，第二个参数是一个Iterator(例如list等)。

其作用，是将函数指针，依次作用到序列的每个元素，并把结果作为新的Iterator返回。

例如:

def f(x):

return x \* x

r = map(f, [1, 2, 3])

print(list(r))

由于结果r是一个Iterator，Iterator是惰性序列，因此通过list()函数让它把整个序列都计算出来并返回一个list。

我们可以利用map做更多复杂的操作，例如：

list( map(str, [1, 2, 3]) )

这里会将数字转换为字符串。

## 4.3.filter函数

filter函数用来过滤序列。

与map类似，filter也接收两个函数：第一个参数是一个函数指针，第二个参数是一个序列。

与map不同的是，filter函数把传入的参数作用于序列中的每个元素，然后根据返回值是True还是False决定保留还是丢弃该元素。

例如：

def isOdd(x):

return x % 2 == 1

list(file(idOdd, [1, 2, 3, 4, 5]))

其输出的list，将是[1, 3, 5]

## 4.4.sorted函数

sorted函数用来排序。

最简单的情况，可以对list进行基本排序：

sorted([3, 5, 2, 1])

高阶使用，我们可以传入一个函数作为key，依照该函数的定义来进行排序，例如：

sorted([3, -5, 2, -1], key=abs)

注意，使用key关键字指明该函数；

key指明的函数，将作用于list的每一个元素上，并根据key返回的结果进行排序。

字符串排序：

sorted([‘adf’, ‘dfet’, ‘AFEGT’], key=str.lower)

通过该key，可以使得排序操作，忽略大小写字母的差异。

反序：

sorted([3, -5, 2, -1], key=abs, reverse=True)

## 4.5.匿名函数

可以使用lambda关键字直接定义一个匿名函数，所谓匿名函数，就是不声明函数名称，直接使用函数过程。

例如：lambda x : x \* x

这个匿名函数，实现的就是平方的功能；

关键字lambda表示匿名函数，紧跟的x表示参数；

匿名函数的限制：只能有一个表达式，不能写return，返回值就是这个表达式的结果；

匿名函数的好处，是无需声明函数，可以减少短小函数带来的命名冲突的可能性；

# 5.模块

python中，一个py文件，称之为一个模块(module)。

模块，可以避免函数名称和变量名称的冲突。

但是，如果不同的开发者使用了相同的模块名字，问题依然存在。

python为了解决该问题，提供了“包”的概念(package)。

一个package，指的是一个特殊的目录，这个目录中，包含了\_\_init\_\_.py文件。这个目录下的所有py文件，都属于这个包了。

## 5.1.模块的标准文件模板

#!/usr/bin/python3

# -\*- coding : utf-8 -\*-

“”” comments for this py “””

\_\_author\_\_ = “WuJinlei”

如上为一个py的标准文件模板，包含了：

1. python解释器；
2. 编码方式；
3. 模板说明；
4. 作者；

## 5.2.private函数/变量

一些函数或变量，只希望本模块内使用，其他模块不能使用，类似于C++中的static或者private，在python中，也可以实现。

python，要限制一个函数或者变量的作用域，在本模块内，方式是对其增加”\_“或者”\_\_“前缀，例如：\_abc, \_\_abc。

# 6.面向对象编程

面向对象编程，Object Oriented Programming, OOP, 将对象作为程序的基本单元，一个对象包含了数据和操作数据的函数。

## 6.1.类和实例

类的声明，使用的关键字是 class，例如：

class Stu(object):

pass

class关键字后紧跟的是类名，括号中表示父类是谁，如果没有合适的父类，就是用object类，这是所有类都会继承的类。

对象可以如此声明：s = Stu()

类的传入参数，使用\_\_init\_\_方法来限定：

class Stu(object):

def \_\_init\_\_(self, a, b):

self.a = a

self.b = b

对象声明就必须修改为：s = Stu(a, b)

这时候，可以直接在对象中访问到a和b的值，例如：s.a=1, s.b=2都是允许的；

限制变量的适用范围，不允许变量直接访问到，对数据进行private处理，可以使用”\_\_”进行处理，例如：

class Stu(object):

def \_\_init\_\_(self, a, b):

self.\_\_a = a

self.\_\_b = b

这时候在使用s.\_\_a = 1就不可以了。

这时候就需要提供get和set方法，来对私有变量进行读取和修改了。

## 6.2.继承和多态

如前所述，继承可以简单实现，例如：

class Animal(object):

pass

class Dog(Animal):

pass

这里dog就是Animal的子类了，可以使用Animal中所有已经定义了的方法了；

如果Animal有一个函数go(),我们在Dog里又声明了这样的同名函数go(),那么Dog中的函数实现，会覆盖Animal中的，这个称为python的多态，类似于virtual的概念。

## 6.3.实例属性和类属性

前述的类中的变量，都是各个对象独自占有的属性，我们还可以对类声明属性，这样所有的变量都可以访问到，例如：

class Stu(object):

count = 0

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

self.\_\_a = a

这里的count就是类属性，所有对象都可以访问到。

# 7.错误、调试和测试

## 7.1.错误

python提供了：try…except…finally…的错误处理机制；

逻辑是：try代码块中的代码，如果抛出了某个异常，并且使用except捕获了这种类型的错误，那么这行代码之后的代码将不会继续执行，而是直接跳转到except中；执行完except后，如果有finally语句，将继续执行finally语句，直至结束。

如果try代码块中有异常，但是没有except这种类型的异常，那么代码会直接断死。

使用try…except机制，最大的好处，是只要调用者在最外层进行了except的调用，就可以捕获到错误，无需每处都进行调用。

我们也可以自己抛出错误。

抛出错误，使用raise语句，例如：

def func(s):

n = int(s)

if n == 0:

raise ValueError(“Input invalid value : %s”, s)

return 10 / n

这样，外部调用的时候，如果传入的s是“0”，就可以捕获到ValueError异常了，例如：

def bar():

try:

foo(“0”)

except ValueError as e :

print(“get it !”)

raise

这里，bar函数首先捕获到了异常，然后又将这个异常继续向上层抛出了。当然，如果bar已经可以足够处理这个异常了，就无需再次抛出了。但如果bar并不知道如何处理这个异常，那么就需要继续抛出，由上层调用者判断如何处理该异常。

## 7.2.单元测试

编写单元测试时，需要编写一个测试类，从 unittest.TestCase 继承，例如：

class testStu(unittest.TestCase):

pass

这个类中，所有以“test”开头的方法名，都认为是需要被执行的单元测试方法；其他方法认为是内部方法，不会再单元测试的时候被执行。

由于unittest.TestCase提供了很多内置的条件判断，我们只需要调用这些方法，就可以断言出输出是否是我们所期望的，例如:

def testAbs(self):

self.assertEqual(abs(-1), 1)

assertEqual断言，用来判断两个值是否相等，为通用的断言函数之一。

此外，还可以通过assertRaises()函数，来判断断言是否正常抛出了，例如:

def testR(self):

with self.assertRaises(ValueError):

value = d.empty

运行单元测试，有两种方法：

1. main函数中，调用：unittest.main()；然后使用python，调用该单元测试脚本，就可以了；
2. Python –m unittest myTest；这里假设我们的py文件，就是myTest.py了。

方法2，可以一次运行多个单元测试项。

单元测试中，有两个特殊的方法：setUp和tearDown，这两个方法，会自动在调用每一个测试方法的前后分别被执行。

假设，我们要测试的是一个数据库的类，每一个测试项，都需要先开启数据库，测试完成后再关闭数据库，那么我们就可以将开启数据库的动作，写入到setUp中，将关闭数据库的动作，写入到tearDown中。

# 8.IO编程

## 8.1.文件读写

python的文件读写，与C/C++类似，都是：open， read/write, close的套路，只不过调用的是python实现了的方法，简述如下：

### 8.1.1.打开文件

使用open函数打开文件，例如：

* f = open("test.txt”, “r”)，以读文件模式打开文件，并返回句柄给f，f可以读写使用；
* f = open(“test.file”, “rb”)，以读模式打开二进制文件；
* f = open(“test.file”, “r”, encoding=”gbk”)，打开的文本文件，并不是utf-8的编码格式，而是gbk编码的，通过encoding参数指定其编码格式并打开；
* f = open(“test.file”, “r”, encoding = “gbk”, errors=”ignore”)，打开的文本文件中，如果遇到了gbk编码中认为是非法编码的字符，处理方式有errors参数指定，这里直接设置为ignore，就是忽略他们了；
* f = open(“test.file”, “w”)，以写模式打开文件，这种模式会清空文件中的原有内容，并执行重新写入的动作，支持wb模式打开二进制文件进行写操作；
* f = open(“test.file”, “a”)，以追加模式打开文件，该模式不会清空文件的原有内容，会直接在文件末尾，开始本次的文件操作，支持wb模式打开二进制文件进行写操作；

如果要打开的文件不存在，或者其他打开文件失败，open会抛出IOError的异常。

### 8.1.2.读文件

使用read函数读取文件，例如：

* s = f.read()，直接将句柄f所打开的文件，所有内容读取到内存中，存储在s中。这个读取方式，只能用在小文件上，文件size过大，会直接将内存耗尽！
* s = f.read(size)，指定size的读取，每次只读取特定大小的文件，这种模式，适合在并不知道文件的确切size的情况下，每次读取指定长度的内容，直到最终；
* s = f.readlines()，将文件内容，按行返回到s中；
* s = f.readline()，读取文件的一行；

for line in f.readlines():

print(line)

### 8.1.3.写文件

使用write函数写文件，例如：f.write(“abc”)

### 8.1.4.关闭文件

关闭文件，是必须要做的操作，否则会造成打开的文件句柄丢失！

f.close()

另外，在我们些数据的时候，操作系统并不是立即将数据更新到磁盘上的，而是放到内存中缓存起来，等到空闲的时候再写入。只有调用close方法的时候，操作系统才将没有写入磁盘的数据，一次性写入其中。

忘记close函数的后果，除了句柄泄露，更可能造成只有一部分数据写到了磁盘中，另一部分就丢失了!

## 8.2.操作文件和目录

python操作文件和目录的函数，一部分在os模块中，一部分在os.path模块中。

* os.path.abspath(“./file”)，查看文件绝对路径；
* os.path.join(“dir”, “file”)，将dir和file，组成一个平台无关的绝对路径，join方法自动将操作系统的分隔符进行设置；
* os.path.split(“filepath”)，将filepath分割成一个元组，前一部分是dir，后一部分是file，其自动判断平台并区分分隔符；
* os.path.splittext(“filepath”)，得到filepath的file的后缀，方法返回一个元组，后一部分是后缀；
* os.mkdir(“dir”)，创建一个目录，如果目录已经存在，将抛出一个FileExistsError；
* os.rmdir(“dir”)，删除一个目录；
* os.rename(“name1”, “name2”)，重命名文件；
* os.remove(“file”)，删除文件；

我们可以利用这些特性，做一些过滤，例如：

* [x for x in os.listdir(“./”) if os.path.isdir(x)]，输出一个list，将所有当前目录下的目录列入其中；
* [x for x in os.listdir(“./”) if os.path.isfile(x) and os.path.splittext(x)[1] = “.py”]，将当前目录下，所有py文件，列入一个list中返回；

# 9.常用内建模块

# 10.常用第三方模块

## 10.1.安装第三方模块

Python中，安装第三方模块，是通过包管理工具pip完成的。

Mac或linux，默认已经安装了pip，只是如果是python3.x的版本，需要选择的是pip3这个可执行程序；windows，安装python时要勾选pip和add python.exe to Path这两个选项，如果勾选了，那么直接在dos中输入pip，将提示pip的基本信息；否则就重装python；

假设我们要安装Pillow库，有两种方式：

* pip install Pillow ： 一般第三方库都会在python官网pypi.python.org注册，可以去该网站搜索第三方库的名字，之后直接执行pip install命令进行安装；
* Anaconda ： 这是一个基于python的数学计算平台，内置了数十个有用的第三方库，安装之后就可以直接使用例如Pillow，NumPy，Flask等第三方模块了；

### 10.1.1.Anaconda在eclipse下的安装使用

已经使用eclipse + pydev搭建了python开发环境。

1. 下载对应的windows的Anaconda安装文件，我的是64位的，因此下载了64为的安装文件；
2. 安装Anaconda；
3. Eclipse，windows--preferences--pydev--interpreter--python interpreter中：
   1. 点击New按钮；
   2. 找到Anaconda中的python.exe文件；
   3. 添加该文件；

安装完成后，可以通过：import numpy 命令，测试一下是否成功。

## 10.2.Pillow

Pillow的前身，是PIL，Python Imaging Library，是python平台事实上的图像处理标准库。其功能非常强大，但是仅仅支持到python 2.7，因此才开发了Pillow，可以支持到python 3.x。

其官网：<https://pillow.readthedocs.org/>

模块名称：PIL，这里要注意，引用的时候，模块名字不是Pillow，而是PIL。

例如，图像缩放：

from PIL import Image #引入模块

im = Image.open(“a.jpg”) #打开图片

width, height = im.size #得到图片现有size

im.thumbnail(width//2, height//2) #缩小图片size

im.save(“smallA.jpg”, “jpeg”) #另存为另一张图片，指定图片格式为jpeg

又比如，模糊效果：

from PIL import Image, ImageFilter #引入模块

im = Image.open(“a.jpg”) #打开图片

Im2 = im.filter(ImageFilter.BLUR) #模糊

Im2.save(“blur.jpg”, “jpeg”) #另存

picamera

该库是在树莓派上进行抓图、录像时用到的库，记录在此。

参阅的是其官方1.13版本的介绍文档，url是: <https://picamera.readthedocs.io/en/release-1.13/>

# 1.安装

如果安装的是树莓派的完整操作系统，那么默认是已经安装了picamera了的，可以通过如下方式查看是否已经安装过了：python3 -c “import picamera”。如果命令执行之后不出现错误提示，说明已经安装了；如果出现“No module named picamera”类似的错误，说明没有安装，要手动执行安装。

sudo apt-get install python3-picamera

有些时候提示安装包找不到，可以通过:apt-get update; apt-get upgrade更新一下软件元信息试试；我这样也提示找不到，之后发现是源的问题，用了阿里的源就更新不到picamera，更改成树莓派原来的源和清华的源，都可以。

需要清除该库，可以:sudo apt-get remove python3-picamera;

# 2.基本使用

要注意，不要为自己的应用程序命名为picamera.py！！因为picamera已经使用了这个文件名字；

引用picamera的方式：from picamera import PiCamera;

使用picamera必须要做的：

with PiCamera() as camera:

camera.start\_preview()

sleep(t) //这两部操作是在为摄像头预热预留时间,t可以设置为秒级的数字

.... //这时候执行自己的操作就可以了

如上的使用方式无需手动释放资源，如果是手动打开对象，理论上还需要释放：

camera = PiCamera()

... //抓图/录像操作

camera.close()

当然，更好的方式是使用 try...finally...实现close；

## 2.1.抓图

### 2.1.1.capture函数原型

抓图由capture方法完成，其函数原型如下：

capture(output, format = None, use\_video\_port=False, resize=None, splitter\_port=0, bayer=False, \*\*options);

该函数完成的功能，就是从摄像头抓取图片，保存在output中；

#### output

抓图的图片的输出对象。

如果是一个字符串，将作为文件名使用，例如：test.jpeg, test.png等，后缀名默认作为图片格式；

如果不是一个字符串，那么必须是一个可以读写的对象，例如ioBytes所指向的一块内存；

#### format

要输出的图片的格式，默认是空，图片的格式由output决定；不为空的时候，可以支持如下一些格式：

* ‘jpeg’：jpeg格式的图片；
* ‘png’：
* ‘gif’：
* ‘bmp’：windows bitmap图片；
* ‘yuv’：yuv420 format；
* ‘rgb’：24-bit rgb format；
* ‘rgba’：32-bit rgba format；
* ‘bgr’：24-bit bgr format；
* ‘bgra’：32-bit bgra format；
* ‘raw’：纯数据模式；不被推荐的模式；

要注意，如果output中指定了输出文件的格式，format中也指定了输出格式，两者不相同的情况下，无法得到正常的图像；

#### use\_video\_port

该参数标记使用摄像头的图像端口还是视频端口进行抓图，默认为false，说明是使用图像端口。

图像端口抓图速度慢，但是图像质量高；视频端口捕获速度快，但那时质量低。如果要使用视频端口，可以将该参数设置为true。

#### resize

设置输出文件的size；

例如对camera设置了resolution为320\*240：camera.resolution = (320, 240)，可以通过该参数修改之，使输出的图片的分辨率改为80\*60：camera.capture(“1.jpg”, resize=(80, 60))

### 2.1.2.抓图到文件

from time import sleep

from picamera import PiCamera

with PiCamera() as camera:

camera.resolution = (320, 240)

camera.start\_preview

for i in range(5):

sleep(1)

camera.capture(str(i) + “.jpg”, resize=(80, 60))

### 2.1.3.抓图到流

可以将抓图直接保存到一个类文件的对象中(一个socket，一个io.BytesIO流，或者一个存在的文件句柄等)：

from io import BytesIO

from time import sleep

from picamera import PiCamera

my\_stream = BytesIO()

with PiCamera() as camera:

camera.start\_preview()

sleep(1)

camera.capture(my\_stream, ‘jpeg’)

要注意，由于流类型不能指定文件类型的后缀，因此必须使用format显示指定输出文件的类型；

### 2.1.4.抓图到PIL

### 2.1.5.consistent images

### 2.1.6.timeelapse sequences

### 2.1.7.low light

### 2.1.8.network stream

## 2.2.录像

### 2.2.1.函数原型

录像所使用的几个函数原型分别介绍。

#### start\_recording

start\_recording(output, format=None, resize=None, splitter\_port=1, \*\*options)

output如果是一个字符串，将作为文件名指定一个要写入的目标文件；其后缀指明输出的视频格式；如果不是一个字符串，就需要是一个可读写的类文件的对象，例如socket、文件句柄、io.BytesIO等；

format支持如下几种赋值：

* ‘h264’:
* ‘mjpeg’:
* ‘yuv’:yuv420 format;
* ‘rgb’:24-bit rgb format;
* ‘rgba’:32-bit rgba format;
* ‘bgr’:24-bit bgr format;
* ‘bgra’:32-bit bgra format;

resize：一个二元的元组，用来指定输出的宽高。

**TODO，按照官网所述这里似乎并不支持h264，要确认一下！**

splitter\_port参数用来指定视频编码器绑定内部的哪个splitter。默认情况下这个值设置成1，大部分的使用者没有任何必要修改这个值。如果需要同时记录多个流(例如以不同的size)，那么这里可以指定一个0--3之间的值来实现。但要注意，一定要制定一个没有被使用的值。

H264格式接受一些额外的参数，以键值对的形式作为输入，也就是options参数的形式：

* profile：默认是“high”，可以被设置成：baseline, main, extended, high, constrained中的一个；
* level：用来编码的H264 level，默认是4，可以设置到4.2；
* intra\_period：
* intra\_refresh
* inline\_headers
* sei：默认是false，表示编码器不需要包含sps信息(supplemental enhancement information)，可以通过指定该值为true要求编码器包含该信息；
* sps\_timing：默认是false，设置为true之后编码器将包含sps heasder；
* motion\_output：

#### wait\_recording

函数原型：wait\_recording(timeout=0, splitter\_port=1)

相当于指定了录像时间长度为timeout指定的时长；

其功能与sleep很相似，但是要注意，它具有sleep不能比的一个功能：录制过程中，如果出现异常导致视频输出失败，该函数是可以上抛异常的！

#### stop\_recording

函数原型：stop\_recording(splitter\_port=1)

### 2.2.2.录像到文件

import picamera

camera = picamera.PiCamera()

camera.resolution = (640, 480)

camera.start\_recording(“test.h264”)

camera.wait\_recording(60)

camera.stop\_recording()

camera.close()

### 2.2.3.录像到流

from io import BytesIO

from picamear import PiCamera

stream = BytesIO()

camera = PiCamera()

camera.solution = (640, 480)

camera.start\_recording(stream, format = “h264”, quality = 23)

camera.wait\_recording(15)

camera.stop\_recording()

### 2.2.4.recording over multiple files

### 2.2.5.recording to a circlar stream

### 2.2.6.recording to a network stream

### 2.2.7.overlaying images on the preview

### 2.2.8.overlaying text on the output

允许叠加最多255个ascii字符到所有输出，包括预览、图片、录像。要实现这个功能，annotate\_text这个属性用的到，例如：

import picamera

import time

camera = picamera.PiCamera()

camera.solution = (640, 480)

camera.framerate = 24

camera.start\_preview()

camera.annotate\_text = “Hello world”

time.sleep(1)

camera.capture(“test.jpg”)

一个很常用的功能，是设置当前时间，叠加到视频中：

import picamera

import datetime as dt

camera = picamera.PiCamera(resolution=(1280, 720), framerate=24)

camera.start\_preview()

camera.annotate\_background = picamera.Color(“black”)

camera.annotate\_text = dt.datetime.now().strftime(“%Y-%m-%d %H:%M:%S”)

camera.start\_recording(“test.h264”)

start = dt.datetime.now()

while(true):

camera.annotate\_text = dt.datetime.now().strftime(“%Y-%m-%d %H:%M:%S”)

camera.wait\_recording(0.2)

camera.stop\_recording()

关于该功能，有如下几个属性要介绍：

* annotate\_background：指定背景色；
* annotate\_foreground：指定前景色；
* annotate\_text：读取的时候，得到的是当前设置的叠加内容，如果没有叠加，返回的是空字符串；设置的时候，将指定的字符串叠加到output中。字符串必须少于255字节，如果长度超长或者包含了不是ASCII字符的内容，将抛出：PiCameraValueError的错误。
* annotate\_text\_size：调整支持的最大的size。用来指定在输出中会显示多少个字符，有效范围是6-160.默认是32个。

### 2.2.9.controlling the LED

# 3.高级特性