题目：多平台音乐下载器

### 完成人：何来胜 学号：SA19225164

### 完成人：张译 学号：SA19225490

### 完成人：高梦天 学号：SA19225144

### 完成人：章征毅 学号：SA19225498

工作量：

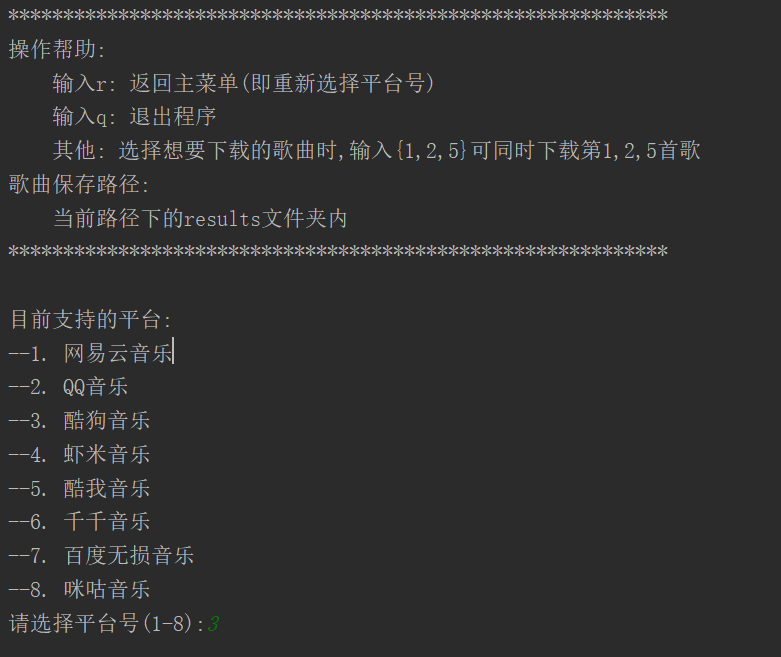
何来胜：30%

张译： 25%

高梦天：25%

章征毅：20%

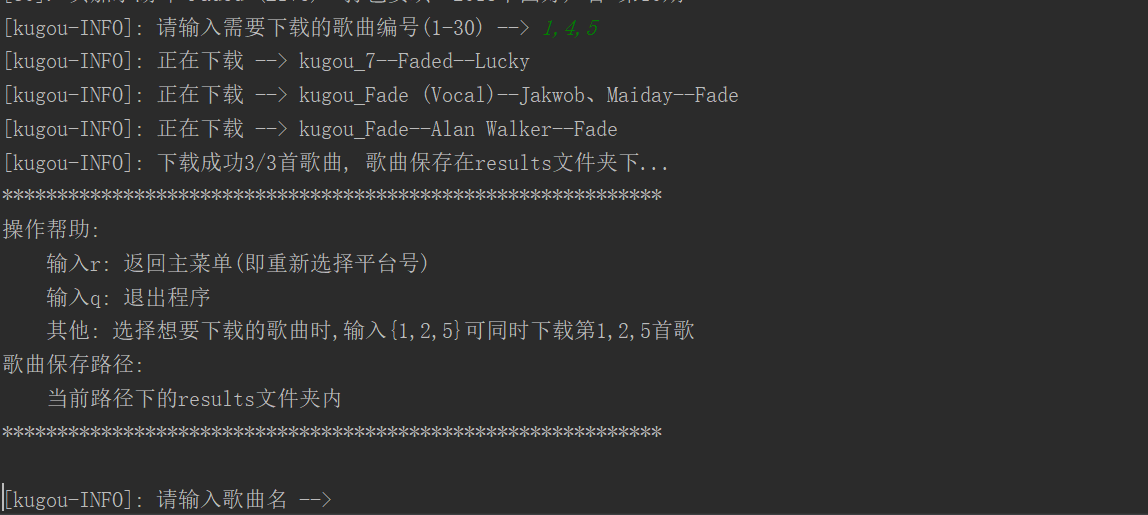
### 一、Demo运行



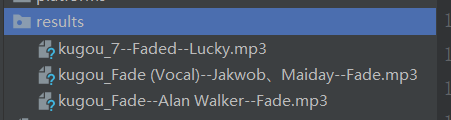
输入平台号之后，输入想要的歌曲名称：



返回搜索结果，并且输入想要下载的歌曲的编号（可多选）：



根据歌曲编号进行下载，并且将下载好的歌曲放入result文件夹中。



### 二、设计背景

近几年，各大音乐平台之间的版权之争愈演愈烈，对于爱听音乐的人来说，有几首歌要在这个平台听，然后又有几首音乐要去另一个平台听是很常见的。而对于下载音乐而言，在电脑上用浏览器打开相应网站通常是无法下载的，各大平台都会要求安装相应的客户端才能下载，而使用手机端下载的音乐通常无法被拷贝到别的设备上听；使用电脑的话，就需要安装各家各户的客户端才能下载音乐，如此多的相同功能的客户端软件，管理起来十分不便，又非常占用空间。因此，我们实现了一个使用了request,re,os,sys,click等库来实现多平台音乐下载的音乐下载器。来解决这个问题，方便的下载音乐。

### 三、设计目标

实现一个可通过输入编号以及歌名实现音乐平台选择，并在相应平台上搜索自己希望下载的音乐名称，然后通过输入指定曲目下载自己想要的音乐的程序。

### 四、技术路线

环境配置：Python3.6

运用到了request,re,os,sys,click等库。

1. 首先，构建构建一个总音乐下载平台类，然后使用面向对象多态的方法调用不同具体音乐平台的类的搜索和下载方法。
2. 使用了request库中的各种方法，如使用了get方法提出请求以及使用re包中的方法获得对应下载的url。

### 五、关键原理

此项目主要由一个总平台类和对应的几个音乐平台类组成，由主平台调用具体平台类的方法，实现了一个面向对象编程中的多态。而且每个类中的方法和成员都设置为私有，防止外部对其直接访问，只提供一个对外部的公由方法接口供外部进行调用。也体现了面向对象编程的封装性的特点。比如在main函数中需要调用cmd类中的run方法来调用cmd类中的私有的搜索和下载方法，然后在这两个方法中调用不同平台的搜索和下载方法，实现下载，实现了一个多态。

Requests 库是用Python语言编写开源的 HTTP 库,实现了 HTTP协议中绝大部分功能。在使用方面极其人性化、简洁。

HTTP的请求类型：

1.GET：它本质就是发送一个请求来取得服务器上的某一资源。资源通过一组HTTP头和呈现数据（如HTML文本，或者图片或者视频等）返回给客户端。GET请求中，永远不会包含呈现数据。

2.POST：向URL指定的资源提交数据或附加新的数据。

3.PUT：通常用于向服务器发送请求，如果URI不存在，则要求服务器根据请求创建资源，如果存在，服务器就接受请求内容，并修改URI资源的原始版本。

4.HEAD:只请求页面首部。

5.DELETE：删除服务器上的某资源。一般很少使用。

6.OPTIONS: 用于获取当前URL所支持的方法。如果请求成功，会有一个Allow的头包含类似“GET,POST”这样的信息。

主要使用的是requests.get() 方法。

它使用HTTP协议中的GET请求方式对目标网站发起请求，同样的还有POST，PUT等请求方式，其中GET是我们最常用的。

requests.get(url, params=None, \*\*kwargs)

url : 拟获取页面的url链接

params : url中的额外参数，字典或字节流格式，可选

\*\*kwargs: 12个控制访问的参数

requests.get()方法返回的是一个包含服务器资源的Response对象。包含从服务器返回的所有的相关资源。

Response对象常用的四个属性：

1.response.status\_code 查看response响应状态

2.response.content

把Response对象的内容以二进制数据的形式返回，适用于图片、音频、视频的下载。

3.response.text

把Response对象的内容以字符串的形式返回，适用于文字、网页源代码的下载

4.response.encoding

帮我们定义Response对象的编码

### 具体实现

总平台类MusicDownloader中，Platform\_now成员中存了当前的具体音乐平台对象，用于实现调用不同平台的搜索和下载方法的一个多态。

self.platform\_now\_name = None

self.is\_select\_platform = False

然后再MusicDownLoader类中有三个私有方法：

def \_\_selectPlatform(self):

def \_\_userSearch(self):

def \_\_download(self, need\_down\_list):

def \_\_input(self, tip=None):

然后再MusicDownloader这个类中有一个共有方法run方法,在这里通过\_\_selectPlatform方法活得了当前选择的具体平台的对象。在selectPlatform方法中调用到了input私有方法。处理输入，已便获得选择的平台。

def run(self):

self.platform\_now, self.platform\_now\_name = self.\_\_selectPlatform()

self.is\_select\_platform = True # 是否选择了平台

在selectPlatForm方法中是输出了当前所有备选的平台以及对应的编号。在其中调用到了input方法，处理用户的输入。并且返回选择到的平台的对象。给类中的成员platform\_now.

while True:

print(self.INFO)

print('目前支持的平台:')

for idx, resource in enumerate(self.RESOURCES):

print('--%d. %s' % ((idx + 1), resource))

platform\_idx = self.\_\_input('请选择平台号(1-%d):' % len(self.RESOURCES))

if platform\_idx == '1':

return wangyiyun.wangyiyun(), 'wangyiyun'

elif platform\_idx == '2':

return qq.qq(), 'qq'

elif platform\_idx == '3':

return kugou.kugou(), 'kugou'

elif platform\_idx == '4':

return xiami.xiami(), 'xiami'

elif platform\_idx == '5':

return kuwo.kuwo(), 'kuwo'

elif platform\_idx == '6':

return qianqian.qianqian(), 'qianqian'

elif platform\_idx == '7':

return baiduFlac.baiduFlac(), 'baiduFlac'

elif platform\_idx == '8':

return migu.migu(), 'migu'

else:

print('<ERROR>--平台号输入有误, 请重新输入--<ERROR>')

在Input方法中，处理用户的输入，判断输入的是哪个平台还是r和q(因为q表示退出程序，r表示返回主菜单(即重新选择平台号))，如果是r和q需要单独处理。表示推出程序或者重新选择。

def \_\_input(self, tip=None):

if tip is None:

user\_input = input()

else:

user\_input = input(tip)

if user\_input.lower() == 'q':

print('Bye...')

sys.exit(-1)

elif user\_input.lower() == 'r':

self.is\_select\_platform = False

if not self.is\_select\_platform:

self.platform\_now, self.platform\_now\_name = self.\_\_selectPlatform()

self.is\_select\_platform = True

return None

else:

return user\_input

Run方法在里面调用了这个类中的私有方法.\_\_userSearch方法。在.\_\_userSearch这个方法当中，首先是获得用户输入的曲名。

songname = self.\_\_input('[%s-INFO]: 请输入歌曲名 --> ' % self.platform\_now\_name)

if songname is None: # 获得歌曲名 并从当前用户输入得平台中搜索获取到歌曲得清单

Return

随后使用到类中的成员platform\_now，里面存了当前的具体平台的对象。然后调用对象的get方法（为具体平台类中提供给外部访问内部私有方法的接口），且mode为搜索。get方法将在后面进行分析，这个方法返回在平台搜索之后得到的对应曲名的清单

results = self.platform\_now.get(mode='search', songname=songname) # 从对应平台中获取到歌曲的清单。

接下来判断result是否为空。

if len(results) == 0:

print('<Warning>--未检索到歌曲%s的相关信息, 请重新输入--<Warning>' % songname)

Return

如果不为空，则要求用户输入要下载的歌曲的曲名。并且做出一些错误判断

print('[%s-INFO]: 搜索结果如下 -->' % self.platform\_now\_name)

for idx, result in enumerate(sorted(results.keys())):

print('[%d]. %s' % (idx + 1, result)) # 显示查询到得歌曲列表清单

need\_down\_numbers = self.\_\_input(

'[%s-INFO]: 请输入需要下载的歌曲编号(1-%d) --> ' % (self.platform\_now\_name, len(results.keys())))

if need\_down\_numbers is None:

return

need\_down\_numbers = need\_down\_numbers.split(',')

numbers\_legal = [str(i) for i in range(1, len(results.keys()) + 1)]

error\_flag = False

for number in need\_down\_numbers: # 判断输入得歌曲编号在输出得查询列表清单中有没有

if number not in numbers\_legal:

print('<ERROR>--歌曲号输入有误, 请重新输入--<ERROR>')

error\_flag = True

break

if error\_flag:

continue

need\_down\_list = []

for number in need\_down\_numbers: # 获得需要下载得歌曲

need\_down\_list.append(sorted(results.keys())[int(number) - 1])

Break

最终获得了一个need\_down\_list列表，里面存着需要下载的歌曲的曲名，然后即可调用MusicDownLoader类中的私有下载方法

downed\_list = self.\_\_download(need\_down\_list) # 调用音乐下载器得download方法

然后\_\_download方法，也是调用了platform\_now成员存着的具体平台的get方法（实现多态），但是mode是下载。

class ParseURL():

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

self.info = 'parse xiami url'

def parse(self, location):

rows, encryptUrl = int(location[:1]), location[1:]

encryptUrlLen = len(encryptUrl)

cols\_base = encryptUrlLen // rows

rows\_ex = encryptUrlLen % rows

matrix = []

for row in range(rows):

length = cols\_base + 1 if row < rows\_ex else cols\_base

matrix.append(encryptUrl[:length])

encryptUrl = encryptUrl[length:]

decryptUrl = ''

for i in range(encryptUrlLen):

decryptUrl += matrix[i % rows][i // rows]

decryptUrl = unquote(decryptUrl).replace('^', '0')

return 'https:' + decryptUrl

如上是获取音乐文件的URL的方法，获取URL的长度后，通过循环将URL复制到matrix中，然后再通过循环将matrix以row为单位拷贝给decryptUrl，最后，剔除掉decryptUrl中的无用字符，加上协议（http）后返回结果。

Get方法是各个具体平台类提供给外部访问内部私有方法的公有接口在这个方法里首先判断mode是search还是download，如果是search,则调用具体音乐平台类中的私有方法\_\_searchBySongname，并且将结果存入search\_results中供后面的download的mode进行使用，可避免重复调用\_\_searchBySongname，然后返回搜索结果。

if mode == 'search':

songname = kwargs.get('songname')

self.search\_results = self.\_\_searchBySongname(songname)

return self.search\_results

如果mode是download，则先从参数中获得需要下载的列表，然后定好下载的歌曲存储的文件夹。

need\_down\_list = kwargs.get('need\_down\_list')

downed\_list = []

savepath = kwargs.get('savepath') if kwargs.get('savepath') is not None else './results'

随后调用在search模式下保存在成员search\_results中搜索结果，由于search\_result中的对象是字典，所以用get方法获得对应的filehash, album\_id。然后利用filehash, album\_id这两个变量的值放入格式化的url字符串中，使用这个url,通过request包的get方法发起get请求，得到返回的报文。通过re包的findall函数返回的正则表达式在字符串中所有匹配结果的列表，即查找在返回报文中play\_url对应的字符串，然后将//替换为’’，即可获得下载的url。随后即可调用具体音乐平台的\_\_download方法进行下载。

elif mode == 'download':

need\_down\_list = kwargs.get('need\_down\_list')

downed\_list = []

savepath = kwargs.get('savepath') if kwargs.get('savepath') is not None else './results'

if need\_down\_list is not None:

for download\_name in need\_down\_list:

filehash, album\_id = self.search\_results.get(download\_name)

res = requests.get(self.hash\_url.format(filehash, album\_id))

play\_url = re.findall('"play\_url":"(.\*?)"', res.text)[0]

download\_url = play\_url.replace("\\", "")

if not download\_url:

continue

res = self.\_\_download(download\_name, download\_url, savepath,

输入平台号之后，输入想要的歌曲名称：

def \_\_searchBySongname(self, songname):

params = {

"key": songname,

"v": "2.0",

"app\_key": "1",

"r": "search/songs",

"page": 1,

"limit": 20,

}

res = self.session.get(self.search\_url, params=params)

results = {}

for song in res.json()['data']['songs']:

if not song.get('listen\_file'):

continue

songid = song.get('song\_id')

singers = song.get('artist\_name')

album = song.get('album\_name')

download\_name = '%s--%s--%s' % (song.get('song\_name'), singers, album)

count = 0

while download\_name in results:

count += 1

download\_name = '%s(%d)--%s--%s' % (song.get('song\_name'), count, singers, album)

results[download\_name] = songid

return results

如上的代码是搜索平台内相应曲目的方法，执行时，传入输入的曲目名称，然后这个曲目名称被保存在param变量内，将param传入类中的get方法，获得相应曲目的URL集，然后通过下方的循环，获得相同名称的不同曲目之间的具体信息，专辑名和乐手名，并将它们存入results中，最后返回results

def \_\_download(self, download\_name, download\_url, savepath, extension):

if not os.path.exists(savepath):

os.mkdir(savepath)

download\_name = download\_name.replace('<', '').replace('>', '').replace('\\', '').replace('/', '') \

.replace('?', '').replace(':', '').replace('"', '').replace('：', '') \

.replace('|', '').replace('？', '').replace('\*', '')

savename = 'xiami\_{}'.format(download\_name)

count = 0

while os.path.isfile(os.path.join(savepath, savename+extension)):

count += 1

savename = 'xiami\_{}\_{}'.format(download\_name, count)

savename += extension

try:

print('[xiami-INFO]: 正在下载 --> %s' % savename.split('.')[0])

with closing(requests.get(download\_url, headers=self.headers, stream=True, verify=False)) as res:

total\_size = int(res.headers['content-length'])

if res.status\_code == 200:

label = '[FileSize]:%0.2f MB' % (total\_size/(1024\*1024))

with click.progressbar(length=total\_size, label=label) as progressbar:

with open(os.path.join(savepath, savename), "wb") as f:

for chunk in res.iter\_content(chunk\_size=1024):

if chunk:

f.write(chunk)

progressbar.update(1024)

else:

raise RuntimeError('Connect error...')

return True

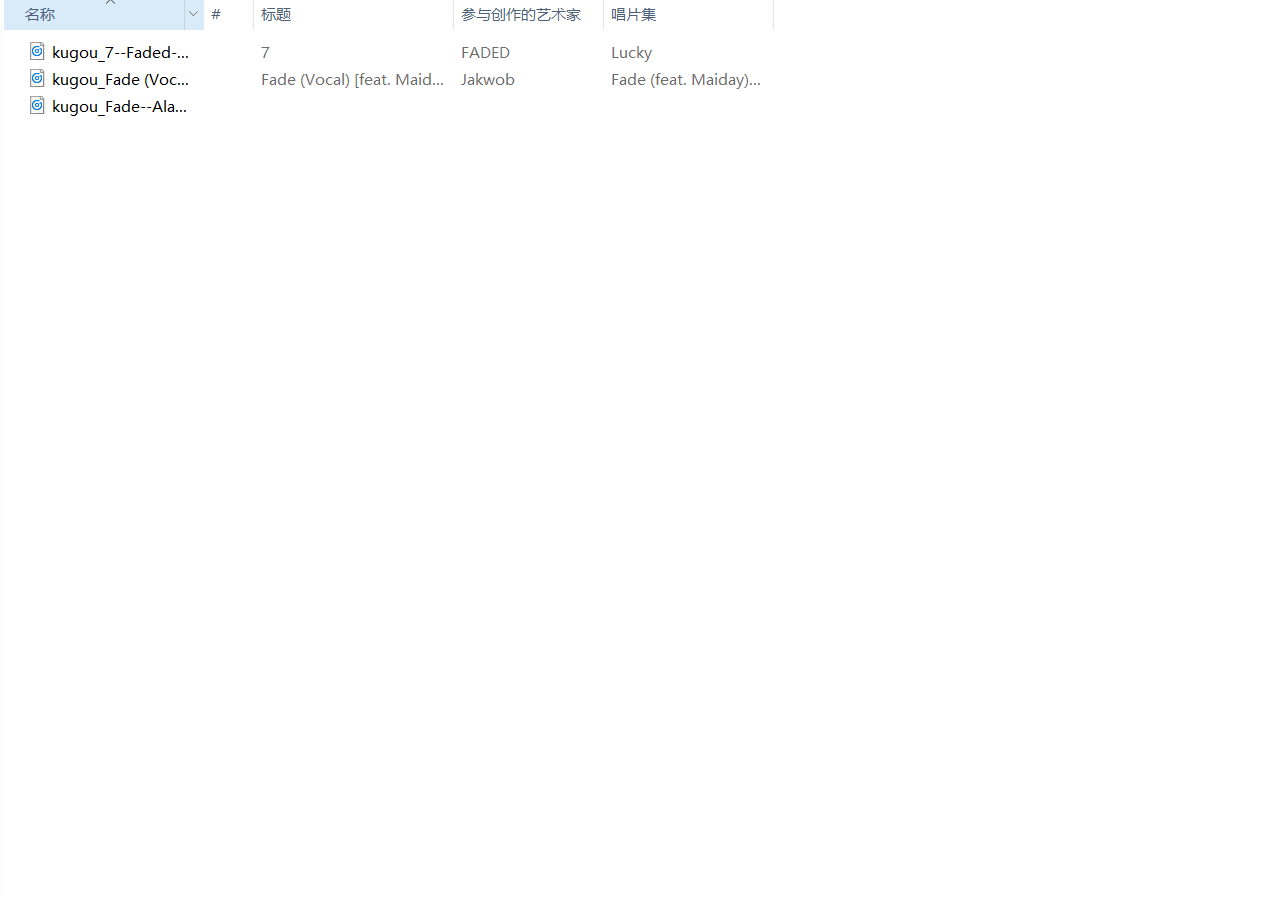
except:

return False

如上的代码是下载平台内相应曲目的方法，此方法接收传入的下载曲目名称，URL，存储路径等信息，将名称里的无效字符剔除掉后，判断是否还有要下载的歌曲，如果有，下载的曲目的计数器自增，然后通过python自带的format字符串格式化函数将将要存储的下载好的曲目名称格式化，随后，曲目名称更新。然后使用python中的with语句代替except和finally，为之后的下载提供异常处理，之后下载曲目并写入存储路径中的相应曲目名称的文件中。

### 七、结果演示

通过操作下载歌曲后，打开相应文件夹



然后打开音乐文件进行播放：



播放成功

### 八、收获和感受

通过这次项目，小组成员们经历了一次从确定项目目标，构思架构，分工合作，最后实现的完整过程。在完成项目的过程中，深刻体会到了python中的面向对象的思想，也实践了使用python中的语法和数据结构以及第三方库进行项目搭建，巩固了对python这门语言的理解。

### 完成时间：2019年11月25日