模拟B站客户端项目报告

**工学院 2021级 通信工程（中外合作办学）于广博 1214023418**

一、项目背景

B站早期是一个ACG内容创作与分享的视频网站。 经过十年多的发展，围绕用户、创作者和内容，构建了一个源源不断产生优质内容的生态系统，B站已经涵盖7000多个兴趣圈层的多元文化社区，曾获得QuestMobile研究院评选的“Z世代偏爱APP”和“Z世代偏爱泛娱乐APP”两项榜单第一名并入选“BrandZ”报告2019最具价值中国品牌100强。

Bilibili早已成为当下最受年轻人欢迎的媒体平台之一，科技区、生活区、游戏区、科普区、时尚区、学习区等各方各面的优秀视频作者时时刻刻为‘小破站’注入新的活力，吸引着广大年轻人在此休闲、学习、娱乐、交流，成为生活中不可或缺的一部分。

二、项目目的

网页端Bilibili不提供视频下载方式。b站有自己的客户端，而客户端下载之后获得m4s文件，普通播放器无法直接播放。移动端的b站客户端提供下载成为MP4文件的方式，然而下载保存后的视频开头和结尾都经过了修改加工，并不能获得原视频，因此有一部分用户有这样的需求：

1.用户想获取B站视频。

2.用户不想要视频片头片尾被有所改动。

3.用户认为在手机端下载再传输到电脑端较为麻烦。

此项目——模拟b站pc端客户端便可解决此类用户的大致需求。

三、项目的可行性

使用chrome的开发者模式观察b站视频播放界面的源代码，发现在视频播放的板块中，存在该视频源的网址链接，这也为爬虫访问该链接获取视频内容提供了可能。同时发现视频的画面和音频是分开存储的，这意味着我们获取出的音画文件是分开的，因此我们可以考虑使用ffmpeg组件来实现对音频和视频的合成。

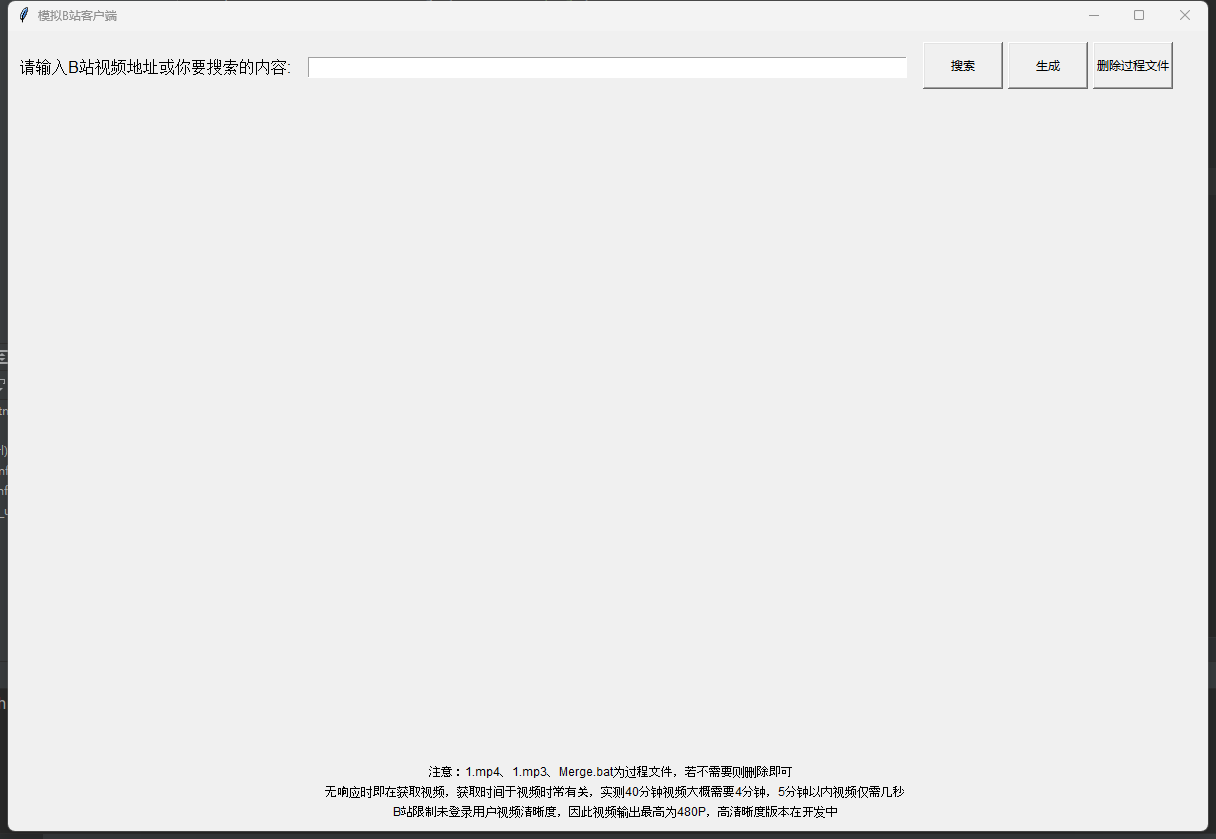


这样获取视频的部分就解决了，而客户端需要一个界面，幸好python有此类的第三方库，此处我可以使用tkinter组件来布置一个窗口。视频标题、封面均在b站网页的源代码中有所呈现，我们只需使用正则表达式在其源代码中筛选出所需信息即可，综上，我们可以实现b站视频的下载功能，也可以实现视频的搜索功能（带着要搜索的内容访问b站的搜索网址），两个共嗯那个配合起来可以算是模拟b站客户端的主要功能了。

四、效果展示

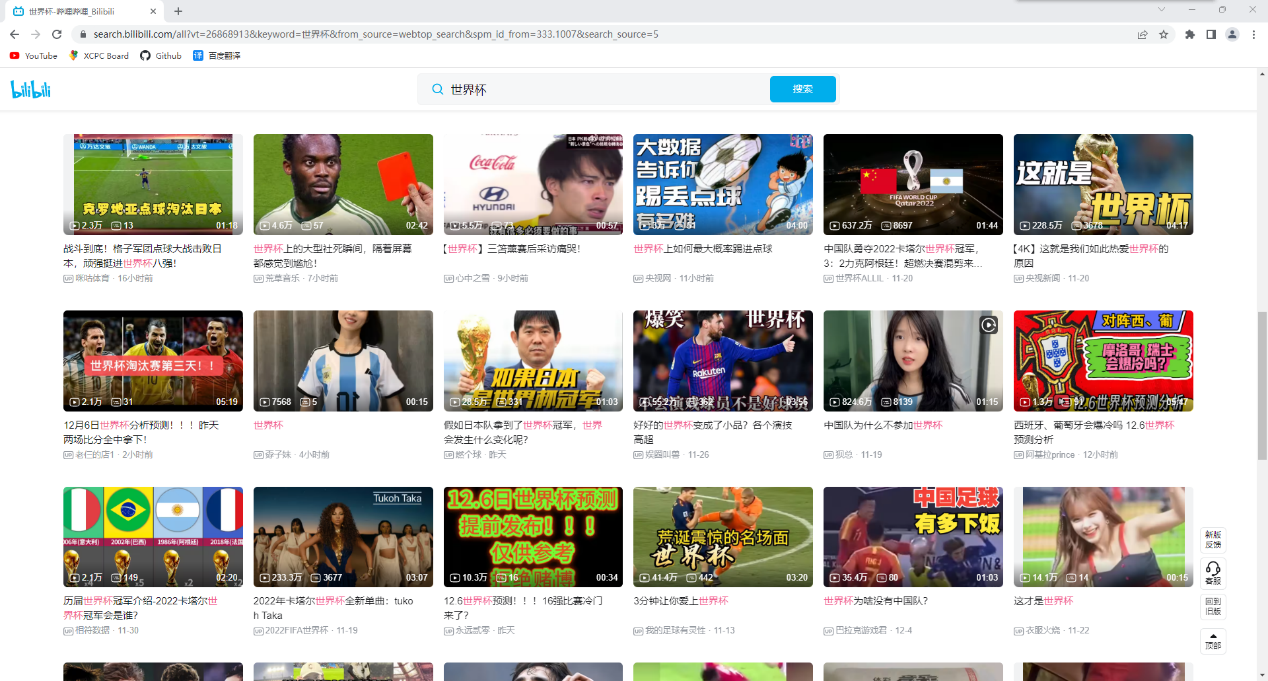
此处以搜索世界杯并下载一个搜索结果的视频为例：

运行程序



文本框中输入世界杯点击搜索（此时会弹出小黑框，那是ffmpeg在处理封面），呈现的六个视频便是在b站网页未登录状态下搜索相同词语所呈现的视频的前六个。



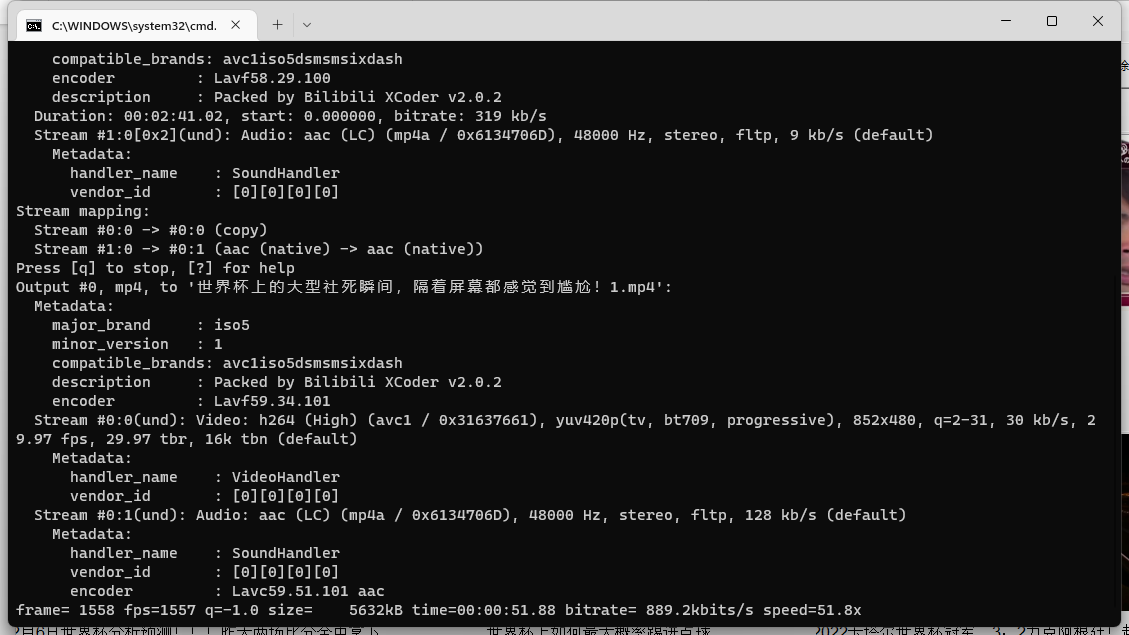


（网页中搜索到的前六个视频便是我们爬取到的六个视频）

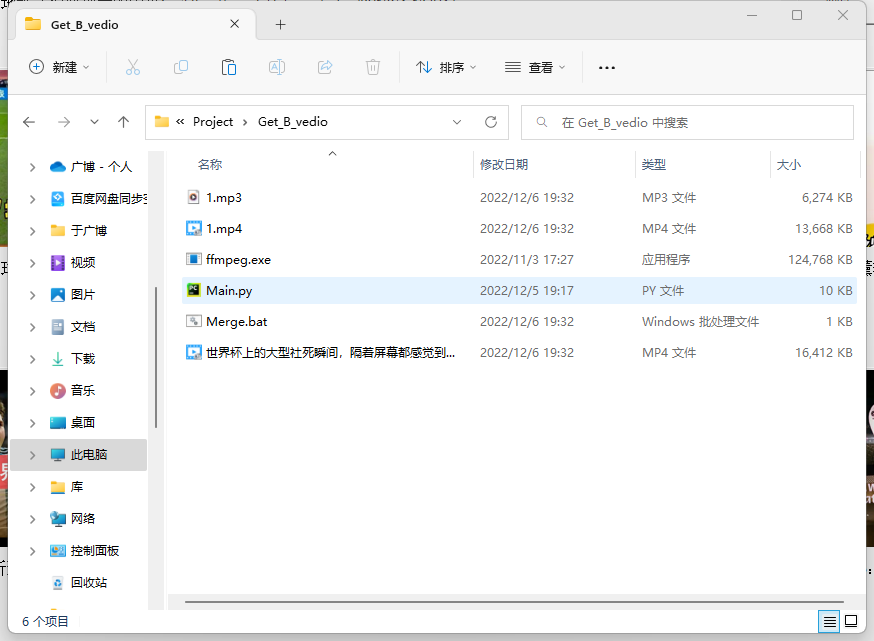
点击视频封面后其链接自动输入到文本框中



点击生成，会产生小黑框，这是ffmpeg在合成视频画面和音频



之后，在程序所在的文件夹就产生了一些文件，其中有1.mp3，1.mp4和想要下载的视频，过程中可能出现一些图片文件以1 2 3 4 11 21 31 41等命名，这是封面的临时文件，点击删除过程文件就可以将其删除。



我们可以看到视频可以正常播放且有声音



另外说明，不使用搜索功能，直接将视频网址输入在文本框当中点击获取也是可以的下载视频的。（指b站视频链接）

五、各部分功能实现

1.所引用的package



requests用于携带信息访问网站或获取网站信息

re用于配合正则表达式对所需信息进行提取

json用于解析网站页面信息，在提取音视频时，json解析后均为字典，提取信息极为方便

subprocess用于执行指令，此项目中用于配置临时环境变量并调用ffmpeg组件实现视频音画合一，并处理视频封面

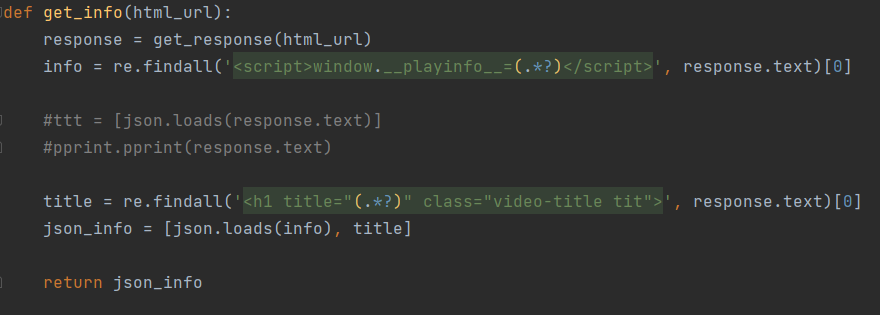
time用于暂停程序，等待组件完成操作

os用于文件操作，视频封面等操作时进行文件读写

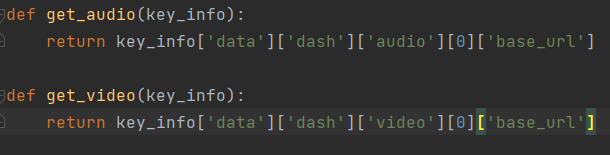
tkinter用于布置窗口

2.各部分功能实现

get\_response函数，主要调用requests库中的get函数获取目标网址的信息，heads此处声明了访问时的来源和设备，用于伪装，避免网站识别出源于python而阻止访问。



get\_info函数，发现在一个bilibili视频播放界面中，主要的音视频源文件链接存放在<script>window.\_\_playinfo\_\_=……</script>这个里面，我们通过正则表达式和re的findall函数获取出来，标题同理，并将这两类信息放在列表中一起返回，这样就获得了这个页面的关键信息。



get\_audio和get\_video用于对get\_info获取之后的关键信息进行提取，由于已经进行json解析，其中都是字典套字典，直接一层一层访问字典的key值就可以获得视频和音频音频的源文件链接

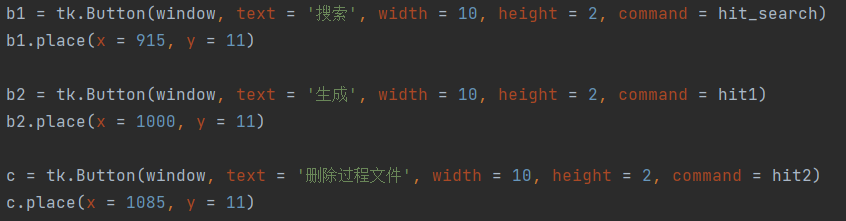


save函数是用来保存的，其中分为好个部分，首先获取音视频内容，将其内容写入1.mp3和1.mp4中，之后处理一下提取出来的标题，并进行重名判定，防止文件重名，之后调用tkinter中的label组件放置在窗口上该视频的信息（窗口的布置后文会讲），最后，将当前文件目录存在loc中，配置该目录的环境变量，并调用ffmpeg组件将1.mp3和1.mp4进行合并，此处通过写一个编辑好命令的bat文件并用os库调用它来实现此过程



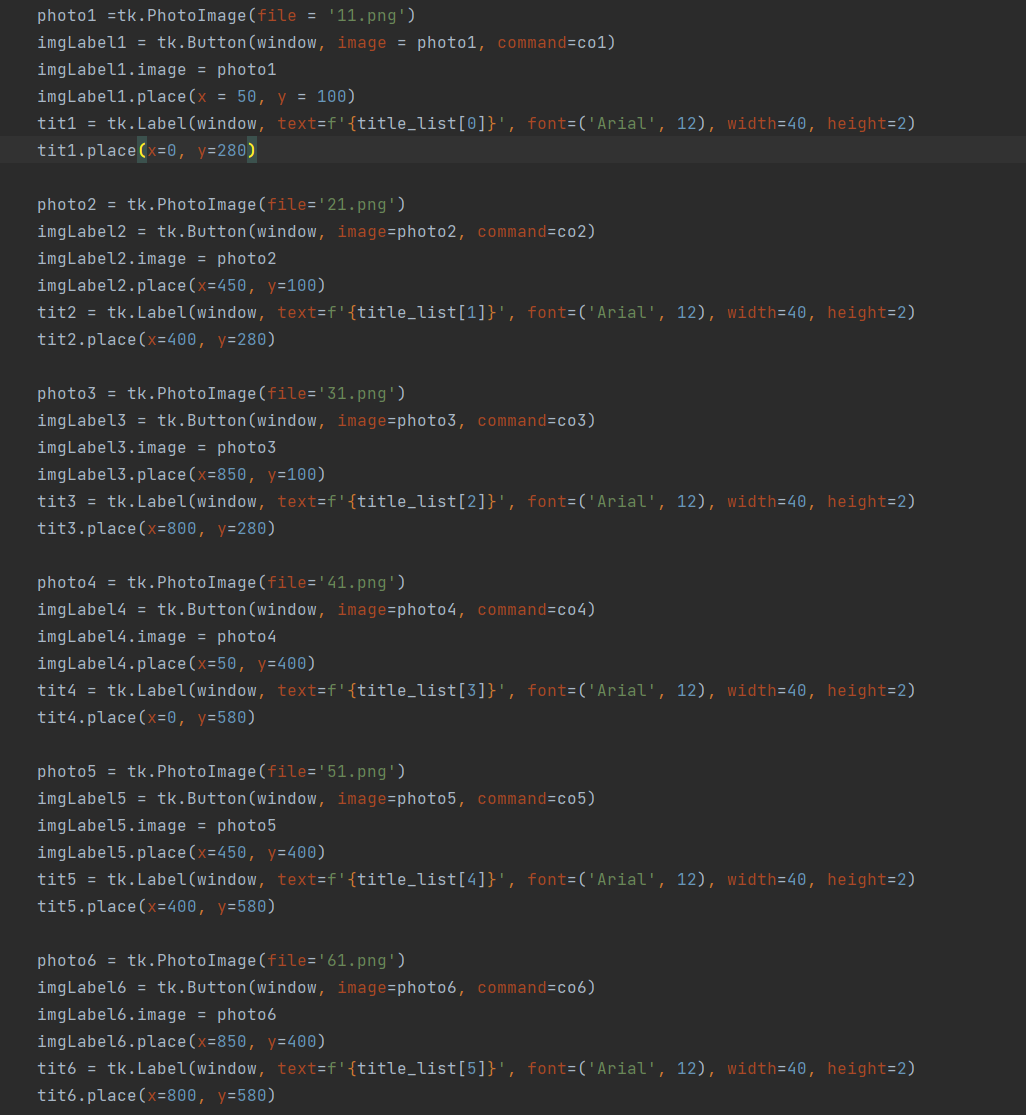
doit函数，实现对文本框中的网址的视频进行爬取并储存，首先确定好目标视频网址，之后调用前面的get\_info函数来获取网页内容，在tkinter的窗口上产生提示语，之后以此获取视频源网址和音频源网址，并送入save函数完成爬取。





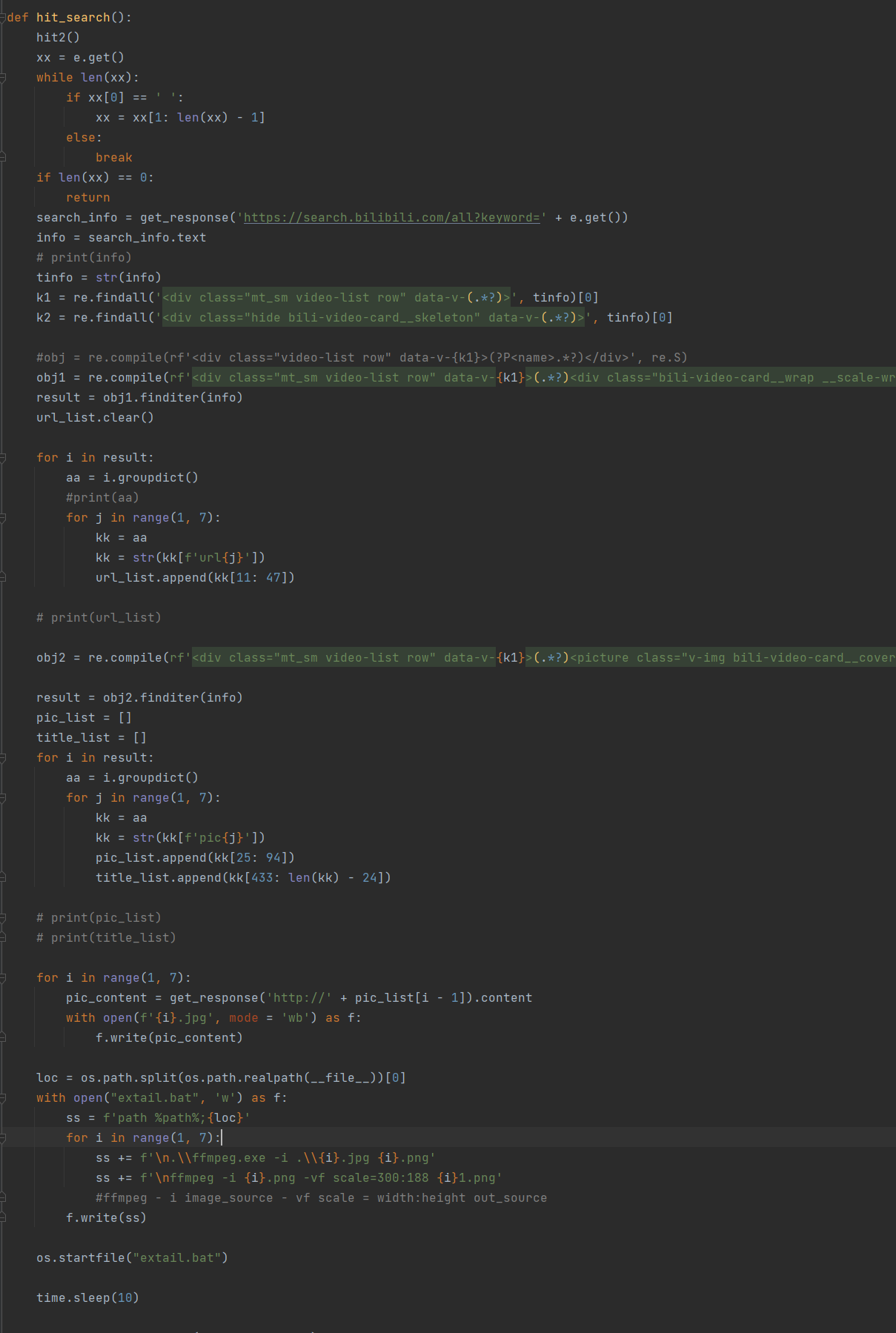
tkinter布置界面，首先生成一个新窗口，起一个名字并规定窗口大小，之后在这个窗口上布置各种小组件，例如标签label，按钮button，文本框entry等等，他们都有自己的参数写在括号里，分别有不同的功能。按钮主要用来触发命定，他触发的命令可以写在一个函数里，放在这条语句的前面即可（类似于函数调用）。标签主要是呈现文本内容，当然也可以有图片。文本框支持编辑输入，可用于与使用者之间交互。

两个hit函数，hit2是在界面中清除过程文件按钮被触发时所执行的函数，就是先判断一下我在合成视频或者封面之后留下的一些过程文件，这些是没有用处的，方便用户清理这些过程文件，也使得文件夹更容易找到生成的用户想要的视频文件，当然也为了文件夹内更美观。此处主要使用os库中的判断文件是否存在和移除文件的指令，要记得在移除文件之前判断一下这个文件是否还存在，若存在我们再执行移除命令。



布置6个带有按钮功能的图片，展示的内容是在b站搜索指定内容后呈现出的前六个视频，其实他们都是按钮，只不过加上了爬取出来的封面图片，他们所对应的指令和如何爬取的视频封面后文有所讲到。之后每个视频封面下面添加了其所对应的视频标题。





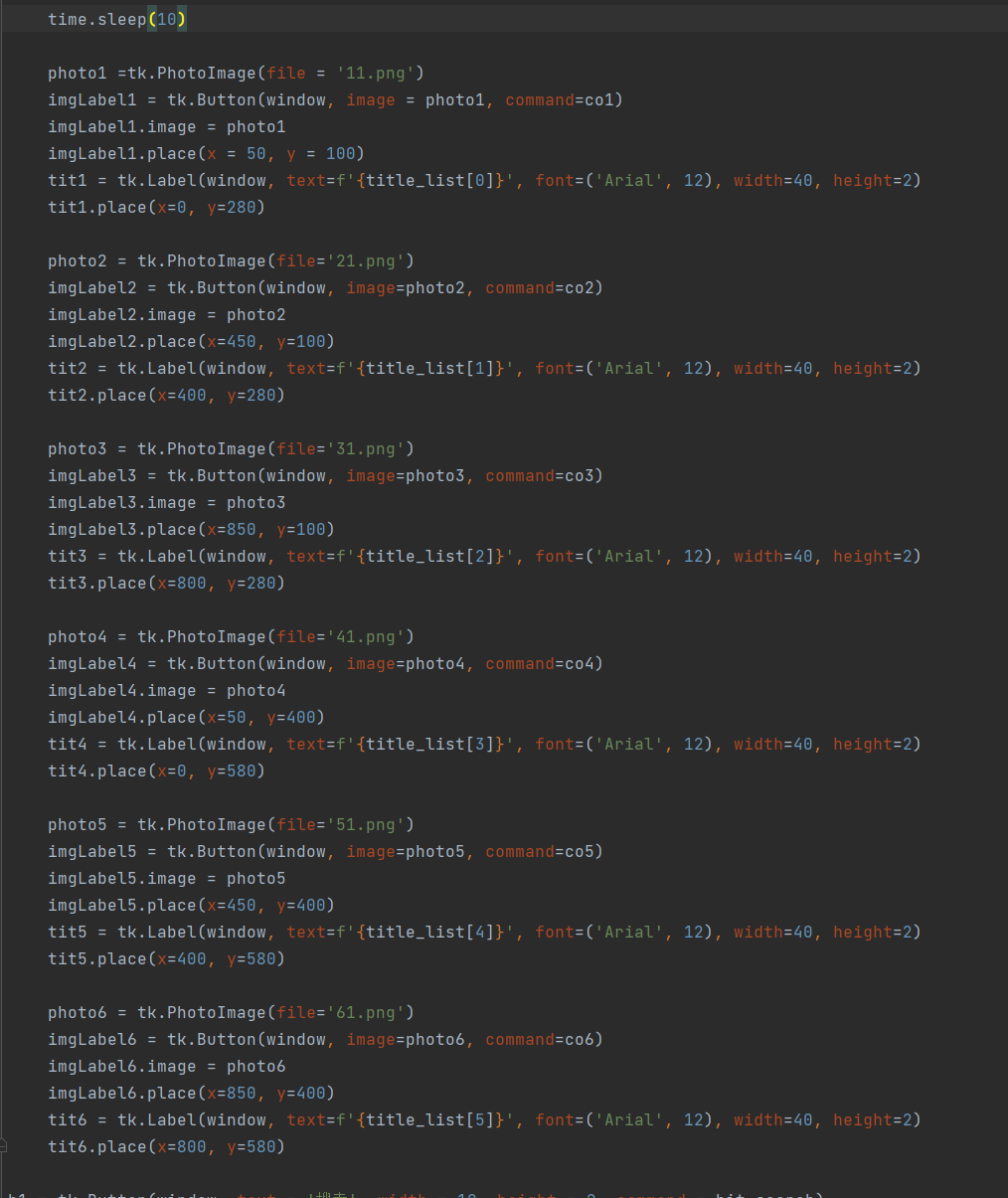
hit\_search函数和生成6个视频标题封面的函数，首先获取文本框中的内容，并对其进行删改，如果什么都没输入或者有前缀空格直接忽略掉。访问bilibili的搜索链接，搜索内容为处理完之后的文本框中要搜索的内容，对于b站搜索该内容的网址，调用之前写好的get\_response函数获取该页信息，提取这一个网页的源代码之后，寻找需要的视频标题和视频封面都是又哪几行代码控制的，图中两行正则表达式太长，具体如下；

obj1 = re.compile(rf'<div class="mt\_sm video-list row" data-v-{k1}>(.\*?)<div class="bili-video-card\_\_wrap \_\_scale-wrap" data-v-{k2}>(?P<url1>.\*?)<div class="bili-video-card\_\_image \_\_scale-player-wrap" data-v-{k2}>(.\*?)<div class="bili-video-card\_\_wrap \_\_scale-wrap" data-v-{k2}>(?P<url2>.\*?)<div class="bili-video-card\_\_image \_\_scale-player-wrap" data-v-{k2}>(.\*?)<div class="bili-video-card\_\_wrap \_\_scale-wrap" data-v-{k2}>(?P<url3>.\*?)<div class="bili-video-card\_\_image \_\_scale-player-wrap" data-v-{k2}>(.\*?)<div class="bili-video-card\_\_wrap \_\_scale-wrap" data-v-{k2}>(?P<url4>.\*?)<div class="bili-video-card\_\_image \_\_scale-player-wrap" data-v-{k2}>(.\*?)<div class="bili-video-card\_\_wrap \_\_scale-wrap" data-v-{k2}>(?P<url5>.\*?)<div class="bili-video-card\_\_image \_\_scale-player-wrap" data-v-{k2}>(.\*?)<div class="bili-video-card\_\_wrap \_\_scale-wrap" data-v-{k2}>(?P<url6>.\*?)<div class="bili-video-card\_\_image \_\_scale-player-wrap" data-v-{k2}>', re.S)

和

obj2 = re.compile(rf'<div class="mt\_sm video-list row" data-v-{k1}>(.\*?)<picture class="v-img bili-video-card\_\_cover" data-v-{k2}>(?P<pic1>.\*?)</picture>(.\*?)<picture class="v-img bili-video-card\_\_cover" data-v-{k2}>(?P<pic2>.\*?)</picture>(.\*?)<picture class="v-img bili-video-card\_\_cover" data-v-{k2}>(?P<pic3>.\*?)</picture>(.\*?)<picture class="v-img bili-video-card\_\_cover" data-v-{k2}>(?P<pic4>.\*?)</picture>(.\*?)<picture class="v-img bili-video-card\_\_cover" data-v-{k2}>(?P<pic5>.\*?)</picture>(.\*?)<picture class="v-img bili-video-card\_\_cover" data-v-{k2}>(?P<pic6>.\*?)</picture>', re.S)

相同种类的信息在网站中是以相同的格式存储的，这为我们写正则表达式提供了很大便利，其实这一堆正则表达式提取六个封面和标题是大部分大部分语句都是相同的，因此其实完全可以写在for循环里进行代码长度的优化。其中正则表达式对大致内容进行提取后仍然不是纯正的标题或者封面源链接，但提取出来的内容在长度上极具规律性，将其转换为字符串在对所需内容对应的字符串下表进行剪切就获得了最终所需的内容。这便是整个信息筛检的思路。之后，在封面源链接获得封面图片式，调用ffmpeg进行对封面的大小的信息进行处理（调用ffmpeg的方式类似于前文视频处理时调用ffmpeg的方式），这样我们就将要搜索的视频的标题和封面都获取到了。（此处只获取了六个视频，其实基于此原理获取多少都是可以的，但是多了就需要有更多的页面来放置，可能还要实现窗口翻页那样的功能，这不是爬虫的范畴，所以没有去实现）。这是搜索功能的主体部分，也是含内容量最多的一部分。



最后一部分：在摁下搜索按钮时，布置六个大按钮，他们的图片分别是六个视频的封面，并在他们下面布置其相应的标题,按下按钮时触发的命定就是co1到co6六个函数，分别将已经获取好的六个视频的链接填写到文本框中，配合之前已经完成的生成功能，将想要的视频进行下载。至此本程序的所有功能及功能的实现思路与方法已全部介绍完毕。

六、缺陷与问题

此项目由于其爬虫功能的实现方式依赖于b站网页的布置，因此有以下几点缺陷与问题。

1．当某天b站视频网页的布置结构发生变化（指与视频、标题、封面、链接等程序中所涉及的网页代码结构），该爬取程序或许无法按原方式完成原信息的爬取。

2．当搜索某个关键词，其对应的视频少于6个时，原程序会报错，因为在提取视频封面与标题时，是按6个在整理好的标题列表和文件夹封面临时文件进行提取，没有对不足六个的情况进行处理。

3.只能搜索/爬取更新为av号之后的b站普通新视频，且必须为普通视频，不能时电影、番剧等，其网页中存储方式有所不同。

4.目前只能下载到480P的分辨率，因为是在未登录的状态下进行爬取，未登陆的状态下b站最高观看到480P分辨率。

六、总解与反思

1．项目初期没有明确规划，做一步看一部导致效率有些低，整个项目包括学习时间大概要有70多小时的样子，之后做项目一定应有明确规划。

2．代码实现随思路变化而写为非先有所规划，导致整体看起来并不是条理清晰，虽然内在逻辑没有大问题，但项目若再庞大一些，势必出现bug后难以调试，此后写成体系的程序时定要率先规划各个板块。

3．对于爬虫的部分，目前的版本虽然已经经过优化，但仍十分依赖于网页的结构布置，因此此程序势必存在时效性，对于爬虫的研究还十分浅显，应虚心求学，道阻且长。