```
#!/usr/bin/env python
# In[ ]:
#  coding: utf-8

###### Searching and Downloading Google Images to the local disk ######

# Import Libraries
import sys
version = (3, 0)
cur_version = sys.version_info
if cur_version >= version:  # If the Current Version of Python is 3.0 or
above
    import urllib.request
    from urllib.request import Request, urlopen
    from urllib.request import URLError, HTTPError
    from urllib.parse import quote
    import http.client
    from http.client import IncompleteRead
    http.client._MAXHEADERS = 1000
else:  # If the Current Version of Python is 2.x
    import urllib2
    from urllib2 import Request, urlopen
    from urllib2 import URLError, HTTPError
    from urllib import quote
    import httplib
    from httplib import IncompleteRead
    httplib._MAXHEADERS = 1000
import time  # Importing the time library to check the time of code execution
import os
import argparse
import ssl
import datetime
import json
import re
import codecs
import socket

args_list = ["keywords", "keywords_from_file", "prefix_keywords",
"suffix_keywords",
             "limit", "format", "color", "color_type", "usage_rights",
"size",
             "exact_size", "aspect_ratio", "type", "time", "time_range",
"delay", "url", "single_image",
             "output_directory", "image_directory", "no_directory", "proxy",
"similar_images", "specific_site",
             "print_urls", "print_size", "print_paths", "metadata",
"extract_metadata", "socket_timeout",
             "thumbnail", "language", "prefix", "chromedriver",
"related_images", "safe_search", "no_numbering",
             "offset", "no_download"]
```

```
def user_input():
    config = argparse.ArgumentParser()
    config.add_argument('-cf', '--config_file', help='config file name',
default='', type=str, required=False)
    config_file_check = config.parse_known_args()
    object_check = vars(config_file_check[0])

    if object_check['config_file'] != '':
        records = []
        json_file = json.load(open(config_file_check[0].config_file))
        for record in range(0,len(json_file['Records'])):
            arguments = {}
            for i in args_list:
                arguments[i] = None
            for key, value in json_file['Records'][record].items():
                arguments[key] = value
            records.append(arguments)
        records_count = len(records)
    else:
        # Taking command line arguments from users
        parser = argparse.ArgumentParser()
        parser.add_argument('-k', '--keywords', help='delimited list input',
type=str, required=False)
        parser.add_argument('-kf', '--keywords_from_file', help='extract list
of keywords from a text file', type=str, required=False)
        parser.add_argument('-sk', '--suffix_keywords', help='comma separated
additional words added after to main keyword', type=str, required=False)
        parser.add_argument('-pk', '--prefix_keywords', help='comma separated
additional words added before main keyword', type=str, required=False)
        parser.add_argument('-l', '--limit', help='delimited list input',
type=str, required=False)
        parser.add_argument('-f', '--format', help='download images with
specific format', type=str, required=False,
                            choices=['jpg', 'gif', 'png', 'bmp', 'svg',
'webp', 'ico'])
        parser.add_argument('-u', '--url', help='search with google image
URL', type=str, required=False)
        parser.add_argument('-x', '--single_image', help='downloading a
single image from URL', type=str, required=False)
        parser.add_argument('-o', '--output_directory', help='download images
in a specific main directory', type=str, required=False)
        parser.add_argument('-i', '--image_directory', help='download images
in a specific sub-directory', type=str, required=False)
        parser.add_argument('-n', '--no_directory', default=False,
help='download images in the main directory but no sub-directory',
action="store_true")
        parser.add_argument('-d', '--delay', help='delay in seconds to wait
between downloading two images', type=int, required=False)
        parser.add_argument('-co', '--color', help='filter on color',
type=str, required=False,
```

```
                            choices=['red', 'orange', 'yellow', 'green',
'teal', 'blue', 'purple', 'pink', 'white', 'gray', 'black', 'brown'])
        parser.add_argument('-ct', '--color_type', help='filter on color',
type=str, required=False,
                            choices=['full-color', 'black-and-white',
'transparent'])
        parser.add_argument('-r', '--usage_rights', help='usage rights',
type=str, required=False,

choices=['labeled-for-reuse-with-modifications','labeled-for-reuse','labeled-
for-noncommercial-reuse-with-modification','labeled-for-nocommercial-reuse'])
        parser.add_argument('-s', '--size', help='image size', type=str,
required=False,

choices=['large','medium','icon','>400*300','>640*480','>800*600','>1024*768'
,'>2MP','>4MP','>6MP','>8MP','>10MP','>12MP','>15MP','>20MP','>40MP','>70MP']
)
        parser.add_argument('-es', '--exact_size', help='exact image
resolution "WIDTH,HEIGHT"', type=str, required=False)
        parser.add_argument('-t', '--type', help='image type', type=str,
required=False,

choices=['face','photo','clipart','line-drawing','animated'])
        parser.add_argument('-w', '--time', help='image age', type=str,
required=False,
                            choices=['past-24-hours','past-7-days'])
        parser.add_argument('-wr', '--time_range', help='time range for the
age of the image. should be in the format
{"time_min":"MM/DD/YYYY","time_max":"MM/DD/YYYY"}', type=str, required=False)
        parser.add_argument('-a', '--aspect_ratio', help='comma separated
additional words added to keywords', type=str, required=False,
                            choices=['tall', 'square', 'wide', 'panoramic'])
        parser.add_argument('-si', '--similar_images', help='downloads images
very similar to the image URL you provide', type=str, required=False)
        parser.add_argument('-ss', '--specific_site', help='downloads images
that are indexed from a specific website', type=str, required=False)
        parser.add_argument('-p', '--print_urls', default=False, help="Print
the URLs of the images", action="store_true")
        parser.add_argument('-ps', '--print_size', default=False, help="Print
the size of the images on disk", action="store_true")
        parser.add_argument('-pp', '--print_paths', default=False,
help="Prints the list of absolute paths of the images",action="store_true")
        parser.add_argument('-m', '--metadata', default=False, help="Print
the metadata of the image", action="store_true")
        parser.add_argument('-e', '--extract_metadata', default=False,
help="Dumps all the logs into a text file", action="store_true")
        parser.add_argument('-st', '--socket_timeout', default=False,
help="Connection timeout waiting for the image to download", type=float)
        parser.add_argument('-th', '--thumbnail', default=False,
help="Downloads image thumbnail along with the actual image",
action="store_true")
```

```
        parser.add_argument('-la', '--language', default=False, help="Defines
the language filter. The search results are authomatically returned in that
language", type=str, required=False,
                            choices=['Arabic','Chinese (Simplified)','Chinese
(Traditional)','Czech','Danish','Dutch','English','Estonian','Finnish','Frenc
h','German','Greek','Hebrew','Hungarian','Icelandic','Italian','Japanese','Ko
rean','Latvian','Lithuanian','Norwegian','Portuguese','Polish','Romanian','Ru
ssian','Spanish','Swedish','Turkish'])
        parser.add_argument('-pr', '--prefix', default=False, help="A word
that you would want to prefix in front of each image name", type=str,
required=False)
        parser.add_argument('-px', '--proxy', help='specify a proxy address
and port', type=str, required=False)
        parser.add_argument('-cd', '--chromedriver', help='specify the path
to chromedriver executable in your local machine', type=str, required=False)
        parser.add_argument('-ri', '--related_images', default=False,
help="Downloads images that are similar to the keyword provided",
action="store_true")
        parser.add_argument('-sa', '--safe_search', default=False,
help="Turns on the safe search filter while searching for images",
action="store_true")
        parser.add_argument('-nn', '--no_numbering', default=False,
help="Allows you to exclude the default numbering of images",
action="store_true")
        parser.add_argument('-of', '--offset', help="Where to start in the
fetched links", type=str, required=False)
        parser.add_argument('-nd', '--no_download', default=False,
help="Prints the URLs of the images and/or thumbnails without downloading
them", action="store_true")

        args = parser.parse_args()
        arguments = vars(args)
        records = []
        records.append(arguments)
    return records


class googleimagesdownload:
    def __init__(self):
        pass

    # Downloading entire Web Document (Raw Page Content)
    def download_page(self,url):
        version = (3, 0)
        cur_version = sys.version_info
        if cur_version >= version:  # If the Current Version of Python is 3.0
or above
            try:
                headers = {}
                headers['User-Agent'] = "Mozilla/5.0 (Windows NT 6.1)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36"
```

```
                req = urllib.request.Request(url, headers=headers)
                resp = urllib.request.urlopen(req)
                respData = str(resp.read())
                return respData
            except Exception as e:
                print("Could not open URL. Please check your internet
connection and/or ssl settings")
        else:  # If the Current Version of Python is 2.x
            try:
                headers = {}
                headers['User-Agent'] = "Mozilla/5.0 (X11; Linux i686)
AppleWebKit/537.17 (KHTML, like Gecko) Chrome/24.0.1312.27 Safari/537.17"
                req = urllib2.Request(url, headers=headers)
                try:
                    response = urllib2.urlopen(req)
                except URLError:  # Handling SSL certificate failed
                    context = ssl._create_unverified_context()
                    response = urlopen(req, context=context)
                page = response.read()
                return page
            except:
                print("Could not open URL. Please check your internet
connection and/or ssl settings")
                return "Page Not found"


    # Download Page for more than 100 images
    def download_extended_page(self,url,chromedriver):
        from selenium import webdriver
        from selenium.webdriver.common.keys import Keys
        if sys.version_info[0] < 3:
            reload(sys)
            sys.setdefaultencoding('utf8')
        options = webdriver.ChromeOptions()
        options.add_argument('--no-sandbox')
        options.add_argument("--headless")

        try:
            browser = webdriver.Chrome(chromedriver, chrome_options=options)
        except Exception as e:
            print("Looks like we cannot locate the path the 'chromedriver'
(use the '--chromedriver' "
                  "argument to specify the path to the executable.) or google
chrome browser is not "
                  "installed on your machine (exception: %s)" % e)
            sys.exit()
        browser.set_window_size(1024, 768)

        # Open the link
        browser.get(url)
        time.sleep(1)
```

```
        print("Getting you a lot of images. This may take a few moments...")

        element = browser.find_element_by_tag_name("body")
        # Scroll down
        for i in range(30):
            element.send_keys(Keys.PAGE_DOWN)
            time.sleep(0.3)

        try:
            browser.find_element_by_id("smb").click()
            for i in range(50):
                element.send_keys(Keys.PAGE_DOWN)
                time.sleep(0.3)  # bot id protection
        except:
            for i in range(10):
                element.send_keys(Keys.PAGE_DOWN)
                time.sleep(0.3)  # bot id protection

        print("Reached end of Page.")
        time.sleep(0.5)

        source = browser.page_source #page source
        #close the browser
        browser.close()

        return source


    #Correcting the escape characters for python2
    def replace_with_byte(self,match):
        return chr(int(match.group(0)[1:], 8))

    def repair(self,brokenjson):
        invalid_escape = re.compile(r'\\[0-7]{1,3}')  # up to 3 digits for
byte values up to FF
        return invalid_escape.sub(self.replace_with_byte, brokenjson)


    # Finding 'Next Image' from the given raw page
    def get_next_tab(self,s):
        start_line = s.find('class="dtviD"')
        if start_line == -1:  # If no links are found then give an error!
            end_quote = 0
            link = "no_tabs"
            return link,'',end_quote
        else:
            start_line = s.find('class="dtviD"')
            start_content = s.find('href="', start_line + 1)
            end_content = s.find('">', start_content + 1)
            url_item = "https://www.google.com" +
str(s[start_content+6:end_content])
```

```
            url_item = url_item.replace('&amp;', '&')

            start_line_2 = s.find('class="dtviD"')
            start_content_2 = s.find(':', start_line_2 + 1)
            end_content_2 = s.find('"', start_content_2 + 1)
            url_item_name = str(s[start_content_2 + 1:end_content_2])

            #print(url_item,url_item_name)
            return url_item,url_item_name,end_content


    # Getting all links with the help of '_images_get_next_image'
    def get_all_tabs(self,page):
        tabs = {}
        while True:
            item,item_name,end_content = self.get_next_tab(page)
            if item == "no_tabs":
                break
            else:
                tabs[item_name] = item  # Append all the links in the list
named 'Links'
                time.sleep(0.1)  # Timer could be used to slow down the
request for image downloads
                page = page[end_content:]
        return tabs


    #Format the object in readable format
    def format_object(self,object):
        formatted_object = {}
        formatted_object['image_format'] = object['ity']
        formatted_object['image_height'] = object['oh']
        formatted_object['image_width'] = object['ow']
        formatted_object['image_link'] = object['ou']
        formatted_object['image_description'] = object['pt']
        formatted_object['image_host'] = object['rh']
        formatted_object['image_source'] = object['ru']
        formatted_object['image_thumbnail_url'] = object['tu']
        return formatted_object


    #function to download single image
    def single_image(self,image_url):
        main_directory = "downloads"
        extensions = (".jpg", ".gif", ".png", ".bmp", ".svg", ".webp",
".ico")
        url = image_url
        try:
            os.makedirs(main_directory)
        except OSError as e:
            if e.errno != 17:
```

```
                raise
            pass
        req = Request(url, headers={
            "User-Agent": "Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.17
(KHTML, like Gecko) Chrome/24.0.1312.27 Safari/537.17"})

        response = urlopen(req, None, 10)
        data = response.read()
        response.close()

        image_name = str(url[(url.rfind('/')) + 1:])
        if '?' in image_name:
            image_name = image_name[:image_name.find('?')]
        # if ".jpg" in image_name or ".gif" in image_name or ".png" in
image_name or ".bmp" in image_name or ".svg" in image_name or ".webp" in
image_name or ".ico" in image_name:
        if any(map(lambda extension: extension in image_name, extensions)):
            file_name = main_directory + "/" + image_name
        else:
            file_name = main_directory + "/" + image_name + ".jpg"
            image_name = image_name + ".jpg"

        try:
            output_file = open(file_name, 'wb')
            output_file.write(data)
            output_file.close()
        except IOError as e:
            raise e
        except OSError as e:
            raise e

        print("completed ====> " + image_name)
        return

    def similar_images(self,similar_images):
        version = (3, 0)
        cur_version = sys.version_info
        if cur_version >= version:  # If the Current Version of Python is 3.0
or above
            try:
                searchUrl =
'https://www.google.com/searchbyimage?site=search&sa=X&image_url=' +
similar_images
                headers = {}
                headers['User-Agent'] = "Mozilla/5.0 (Windows NT 6.1)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36"

                req1 = urllib.request.Request(searchUrl, headers=headers)
                resp1 = urllib.request.urlopen(req1)
                content = str(resp1.read())
                l1 = content.find('AMhZZ')
```

```
                l2 = content.find('&', l1)
                urll = content[l1:l2]

                newurl = "https://www.google.com/search?tbs=sbi:" + urll +
"&site=search&sa=X"
                req2 = urllib.request.Request(newurl, headers=headers)
                resp2 = urllib.request.urlopen(req2)
                # print(resp2.read())
                l3 = content.find('/search?sa=X&amp;q=')
                l4 = content.find(';', l3 + 19)
                urll2 = content[l3 + 19:l4]
                return urll2
            except:
                return "Cloud not connect to Google Images endpoint"
        else:  # If the Current Version of Python is 2.x
            try:
                searchUrl =
'https://www.google.com/searchbyimage?site=search&sa=X&image_url=' +
similar_images
                headers = {}
                headers['User-Agent'] = "Mozilla/5.0 (X11; Linux i686)
AppleWebKit/537.17 (KHTML, like Gecko) Chrome/24.0.1312.27 Safari/537.17"

                req1 = urllib2.Request(searchUrl, headers=headers)
                resp1 = urllib2.urlopen(req1)
                content = str(resp1.read())
                l1 = content.find('AMhZZ')
                l2 = content.find('&', l1)
                urll = content[l1:l2]

                newurl = "https://www.google.com/search?tbs=sbi:" + urll +
"&site=search&sa=X"
                #print newurl
                req2 = urllib2.Request(newurl, headers=headers)
                resp2 = urllib2.urlopen(req2)
                # print(resp2.read())
                l3 = content.find('/search?sa=X&amp;q=')
                l4 = content.find(';', l3 + 19)
                urll2 = content[l3 + 19:l4]
                return(urll2)
            except:
                return "Cloud not connect to Google Images endpoint"

    #Building URL parameters
    def build_url_parameters(self,arguments):
        if arguments['language']:
            lang = "&lr="
            lang_param = {"Arabic":"lang_ar","Chinese
(Simplified)":"lang_zh-CN","Chinese
(Traditional)":"lang_zh-TW","Czech":"lang_cs","Danish":"lang_da","Dutch":"lan
g_nl","English":"lang_en","Estonian":"lang_et","Finnish":"lang_fi","French":"
```

```
lang_fr","German":"lang_de","Greek":"lang_el","Hebrew":"lang_iw
","Hungarian":"lang_hu","Icelandic":"lang_is","Italian":"lang_it","Japanese":
"lang_ja","Korean":"lang_ko","Latvian":"lang_lv","Lithuanian":"lang_lt","Norw
egian":"lang_no","Portuguese":"lang_pt","Polish":"lang_pl","Romanian":"lang_r
o","Russian":"lang_ru","Spanish":"lang_es","Swedish":"lang_sv","Turkish":"lan
g_tr"}
            lang_url = lang+lang_param[arguments['language']]
        else:
            lang_url = ''

        if arguments['time_range']:
            json_acceptable_string = arguments['time_range'].replace("'",
"\"")
            d = json.loads(json_acceptable_string)
            time_range = ',cdr:1,cd_min:' + d['time_min'] + ',cd_max:' +
d['time_max']
        else:
            time_range = ''

        if arguments['exact_size']:
            size_array = [x.strip() for x in
arguments['exact_size'].split(',')]
            exact_size = ",isz:ex,iszw:" + str(size_array[0]) + ",iszh:" +
str(size_array[1])
        else:
            exact_size = ''

        built_url = "&tbs="
        counter = 0
        params = {'color':[arguments['color'],{'red':'ic:specific,isc:red',
'orange':'ic:specific,isc:orange', 'yellow':'ic:specific,isc:yellow',
'green':'ic:specific,isc:green', 'teal':'ic:specific,isc:teel',
'blue':'ic:specific,isc:blue', 'purple':'ic:specific,isc:purple',
'pink':'ic:specific,isc:pink', 'white':'ic:specific,isc:white',
'gray':'ic:specific,isc:gray', 'black':'ic:specific,isc:black',
'brown':'ic:specific,isc:brown'}],

'color_type':[arguments['color_type'],{'full-color':'ic:color',
'black-and-white':'ic:gray','transparent':'ic:trans'}],

'usage_rights':[arguments['usage_rights'],{'labeled-for-reuse-with-modificati
ons':'sur:fmc','labeled-for-reuse':'sur:fc','labeled-for-noncommercial-reuse-
with-modification':'sur:fm','labeled-for-nocommercial-reuse':'sur:f'}],

'size':[arguments['size'],{'large':'isz:l','medium':'isz:m','icon':'isz:i','>
400*300':'isz:lt,islt:qsvga','>640*480':'isz:lt,islt:vga','>800*600':'isz:lt,
islt:svga','>1024*768':'visz:lt,islt:xga','>2MP':'isz:lt,islt:2mp','>4MP':'is
z:lt,islt:4mp','>6MP':'isz:lt,islt:6mp','>8MP':'isz:lt,islt:8mp','>10MP':'isz
:lt,islt:10mp','>12MP':'isz:lt,islt:12mp','>15MP':'isz:lt,islt:15mp','>20MP':
'isz:lt,islt:20mp','>40MP':'isz:lt,islt:40mp','>70MP':'isz:lt,islt:70mp'}],
```

```
'type':[arguments['type'],{'face':'itp:face','photo':'itp:photo','clipart':'i
tp:clipart','line-drawing':'itp:lineart','animated':'itp:animated'}],

'time':[arguments['time'],{'past-24-hours':'qdr:d','past-7-days':'qdr:w'}],

'aspect_ratio':[arguments['aspect_ratio'],{'tall':'iar:t','square':'iar:s','w
ide':'iar:w','panoramic':'iar:xw'}],

'format':[arguments['format'],{'jpg':'ift:jpg','gif':'ift:gif','png':'ift:png
','bmp':'ift:bmp','svg':'ift:svg','webp':'webp','ico':'ift:ico'}]}
        for key, value in params.items():
            if value[0] is not None:
                ext_param = value[1][value[0]]
                # counter will tell if it is first param added or not
                if counter == 0:
                    # add it to the built url
                    built_url = built_url + ext_param
                    counter += 1
                else:
                    built_url = built_url + ',' + ext_param
                    counter += 1
        built_url = lang_url+built_url+exact_size+time_range
        return built_url


    #building main search URL
    def
build_search_url(self,search_term,params,url,similar_images,specific_site,saf
e_search):
        #check safe_search
        safe_search_string = "&safe=active"
        # check the args and choose the URL
        if url:
            url = url
        elif similar_images:
            print(similar_images)
            keywordem = self.similar_images(similar_images)
            url = 'https://www.google.com/search?q=' + keywordem +
'&espv=2&biw=1366&bih=667&site=webhp&source=lnms&tbm=isch&sa=X&ei=XosDVaCXD8T
asATItgE&ved=0CAcQ_AUoAg'
        elif specific_site:
            url = 'https://www.google.com/search?q=' + quote(
                search_term) + '&as_sitesearch=' + specific_site +
'&espv=2&biw=1366&bih=667&site=webhp&source=lnms&tbm=isch' + params +
'&sa=X&ei=XosDVaCXD8TasATItgE&ved=0CAcQ_AUoAg'
        else:
            url = 'https://www.google.com/search?q=' + quote(
                search_term) +
'&espv=2&biw=1366&bih=667&site=webhp&source=lnms&tbm=isch' + params +
'&sa=X&ei=XosDVaCXD8TasATItgE&ved=0CAcQ_AUoAg'
```

```
        #safe search check
        if safe_search:
            url = url + safe_search_string

        # print(url)
        return url


    #measures the file size
    def file_size(self,file_path):
        if os.path.isfile(file_path):
            file_info = os.stat(file_path)
            size = file_info.st_size
            for x in ['bytes', 'KB', 'MB', 'GB', 'TB']:
                if size < 1024.0:
                    return "%3.1f %s" % (size, x)
                size /= 1024.0
            return size

    #keywords from file
    def keywords_from_file(self,file_name):
        search_keyword = []
        with codecs.open(file_name, 'r', encoding='utf-8-sig') as f:
            if '.csv' in file_name:
                for line in f:
                    if line in ['\n', '\r\n']:
                        pass
                    else:
                        search_keyword.append(line.replace('\n',
'').replace('\r', ''))
            elif '.txt' in file_name:
                for line in f:
                    if line in ['\n', '\r\n']:
                        pass
                    else:
                        search_keyword.append(line.replace('\n',
'').replace('\r', ''))
            else:
                print("Invalid file type: Valid file types are either .txt or
.csv \n"
                      "exiting...")
                sys.exit()
        return search_keyword

    # make directories
    def create_directories(self,main_directory, dir_name,thumbnail):
        dir_name_thumbnail = dir_name + " - thumbnail"
        # make a search keyword  directory
        try:
            if not os.path.exists(main_directory):
                os.makedirs(main_directory)
```

```
                time.sleep(0.2)
                path = str(dir_name)
                sub_directory = os.path.join(main_directory, path)
                if not os.path.exists(sub_directory):
                    os.makedirs(sub_directory)
                if thumbnail:
                    sub_directory_thumbnail = os.path.join(main_directory,
dir_name_thumbnail)
                    if not os.path.exists(sub_directory_thumbnail):
                        os.makedirs(sub_directory_thumbnail)
            else:
                path = str(dir_name)
                sub_directory = os.path.join(main_directory, path)
                if not os.path.exists(sub_directory):
                    os.makedirs(sub_directory)
                if thumbnail:
                    sub_directory_thumbnail = os.path.join(main_directory,
dir_name_thumbnail)
                    if not os.path.exists(sub_directory_thumbnail):
                        os.makedirs(sub_directory_thumbnail)
        except OSError as e:
            if e.errno != 17:
                raise
                # time.sleep might help here
            pass
        return


    # Download Images
    def
download_image_thumbnail(self,image_url,main_directory,dir_name,return_image_
name,print_urls,socket_timeout,print_size,no_download):
        if print_urls or no_download:
            print("Image URL: " + image_url)
        if no_download:
            return "success","Printed url without downloading"
        try:
            req = Request(image_url, headers={
                "User-Agent": "Mozilla/5.0 (X11; Linux i686)
AppleWebKit/537.17 (KHTML, like Gecko) Chrome/24.0.1312.27 Safari/537.17"})
            try:
                # timeout time to download an image
                if socket_timeout:
                    timeout = float(socket_timeout)
                else:
                    timeout = 10

                response = urlopen(req, None, timeout)
                data = response.read()
                response.close()
```

```
                path = main_directory + "/" + dir_name + " - thumbnail" + "/"
+ return_image_name

                try:
                    output_file = open(path, 'wb')
                    output_file.write(data)
                    output_file.close()
                except OSError as e:
                    download_status = 'fail'
                    download_message = "OSError on an image...trying next
one..." + " Error: " + str(e)
                except IOError as e:
                    download_status = 'fail'
                    download_message = "IOError on an image...trying next
one..." + " Error: " + str(e)

                download_status = 'success'
                download_message = "Completed Image Thumbnail ====> " +
return_image_name

                # image size parameter
                if print_size:
                    print("Image Size: " + str(self.file_size(path)))

            except UnicodeEncodeError as e:
                download_status = 'fail'
                download_message = "UnicodeEncodeError on an image...trying
next one..." + " Error: " + str(e)

        except HTTPError as e:  # If there is any HTTPError
            download_status = 'fail'
            download_message = "HTTPError on an image...trying next one..." +
" Error: " + str(e)

        except URLError as e:
            download_status = 'fail'
            download_message = "URLError on an image...trying next one..." +
" Error: " + str(e)

        except ssl.CertificateError as e:
            download_status = 'fail'
            download_message = "CertificateError on an image...trying next
one..." + " Error: " + str(e)

        except IOError as e:  # If there is any IOError
            download_status = 'fail'
            download_message = "IOError on an image...trying next one..." + "
Error: " + str(e)
        return download_status, download_message
```

```
    # Download Images
    def
download_image(self,image_url,image_format,main_directory,dir_name,count,prin
t_urls,socket_timeout,prefix,print_size,no_numbering,no_download):
        if print_urls or no_download:
            print("Image URL: " + image_url)
        if no_download:
            return "success","Printed url without downloading",None,None
        try:
            req = Request(image_url, headers={
                "User-Agent": "Mozilla/5.0 (X11; Linux i686)
AppleWebKit/537.17 (KHTML, like Gecko) Chrome/24.0.1312.27 Safari/537.17"})
            try:
                # timeout time to download an image
                if socket_timeout:
                    timeout = float(socket_timeout)
                else:
                    timeout = 10

                response = urlopen(req, None, timeout)
                data = response.read()
                response.close()

                # keep everything after the last '/'
                image_name = str(image_url[(image_url.rfind('/')) + 1:])
                image_name = image_name.lower()
                # if no extension then add it
                # remove everything after the image name
                if image_format == "":
                    image_name = image_name + "." + "jpg"
                elif image_format == "jpeg":
                    image_name = image_name[:image_name.find(image_format) +
4]
                else:
                    image_name = image_name[:image_name.find(image_format) +
3]

                # prefix name in image
                if prefix:
                    prefix = prefix + " "
                else:
                    prefix = ''

                if no_numbering:
                    path = main_directory + "/" + dir_name + "/" + prefix +
image_name
                else:
                    path = main_directory + "/" + dir_name + "/" + prefix +
str(count) + ". " + image_name

                try:
```

```
                    output_file = open(path, 'wb')
                    output_file.write(data)
                    output_file.close()
                    absolute_path = os.path.abspath(path)
                except OSError as e:
                    download_status = 'fail'
                    download_message = "OSError on an image...trying next
one..." + " Error: " + str(e)
                    return_image_name = ''
                    absolute_path = ''

                #return image name back to calling method to use it for
thumbnail downloads
                download_status = 'success'
                download_message = "Completed Image ====> " + prefix +
str(count) + ". " + image_name
                return_image_name = prefix + str(count) + ". " + image_name

                # image size parameter
                if print_size:
                    print("Image Size: " + str(self.file_size(path)))

            except UnicodeEncodeError as e:
                download_status = 'fail'
                download_message = "UnicodeEncodeError on an image...trying
next one..." + " Error: " + str(e)
                return_image_name = ''
                absolute_path = ''

            except URLError as e:
                download_status = 'fail'
                download_message = "URLError on an image...trying next
one..." + " Error: " + str(e)
                return_image_name = ''
                absolute_path = ''

        except HTTPError as e:  # If there is any HTTPError
            download_status = 'fail'
            download_message = "HTTPError on an image...trying next one..." +
" Error: " + str(e)
            return_image_name = ''
            absolute_path = ''

        except URLError as e:
            download_status = 'fail'
            download_message = "URLError on an image...trying next one..." +
" Error: " + str(e)
            return_image_name = ''
            absolute_path = ''

        except ssl.CertificateError as e:
```

```
            download_status = 'fail'
            download_message = "CertificateError on an image...trying next
one..." + " Error: " + str(e)
            return_image_name = ''
            absolute_path = ''

        except IOError as e:  # If there is any IOError
            download_status = 'fail'
            download_message = "IOError on an image...trying next one..." + "
Error: " + str(e)
            return_image_name = ''
            absolute_path = ''

        except IncompleteRead as e:
            download_status = 'fail'
            download_message = "IncompleteReadError on an image...trying next
one..." + " Error: " + str(e)
            return_image_name = ''
            absolute_path = ''

        return
download_status,download_message,return_image_name,absolute_path


    # Finding 'Next Image' from the given raw page
    def _get_next_item(self,s):
        start_line = s.find('rg_meta notranslate')
        if start_line == -1:  # If no links are found then give an error!
            end_quote = 0
            link = "no_links"
            return link, end_quote
        else:
            start_line = s.find('class="rg_meta notranslate">')
            start_object = s.find('{', start_line + 1)
            end_object = s.find('</div>', start_object + 1)
            object_raw = str(s[start_object:end_object])
            #remove escape characters based on python version
            version = (3, 0)
            cur_version = sys.version_info
            if cur_version >= version: #python3
                try:
                    object_decode = bytes(object_raw,
"utf-8").decode("unicode_escape")
                    final_object = json.loads(object_decode)
                except:
                    final_object = ""
            else:  #python2
                try:
                    final_object = (json.loads(self.repair(object_raw)))
                except:
                    final_object = ""
```

```
            return final_object, end_object


    # Getting all links with the help of '_images_get_next_image'
    def _get_all_items(self,page,main_directory,dir_name,limit,arguments):
        items = []
        abs_path = []
        errorCount = 0
        i = 0
        count = 1
        while count < limit+1:
            object, end_content = self._get_next_item(page)
            if object == "no_links":
                break
            elif object == "":
                page = page[end_content:]
            elif arguments['offset'] and count < int(arguments['offset']):
                    count += 1
                    page = page[end_content:]
            else:
                #format the item for readability
                object = self.format_object(object)
                if arguments['metadata']:
                    print("\nImage Metadata: " + str(object))

                #download the images

download_status,download_message,return_image_name,absolute_path =
self.download_image(object['image_link'],object['image_format'],main_director
y,dir_name,count,arguments['print_urls'],arguments['socket_timeout'],argument
s['prefix'],arguments['print_size'],arguments['no_numbering'],arguments['no_d
ownload'])
                print(download_message)
                if download_status == "success":

                    # download image_thumbnails
                    if arguments['thumbnail']:
                        download_status, download_message_thumbnail =
self.download_image_thumbnail(object['image_thumbnail_url'],main_directory,di
r_name,return_image_name,arguments['print_urls'],arguments['socket_timeout'],
arguments['print_size'],arguments['no_download'])
                        print(download_message_thumbnail)

                    count += 1
                    object['image_filename'] = return_image_name
                    items.append(object)  # Append all the links in the list
named 'Links'
                    abs_path.append(absolute_path)
                else:
                    errorCount += 1
```

```
                #delay param
                if arguments['delay']:
                    time.sleep(int(arguments['delay']))

                page = page[end_content:]
            i += 1
        if count < limit:
            print("\n\nUnfortunately all " + str(
                limit) + " could not be downloaded because some images were
not downloadable. " + str(
                count-1) + " is all we got for this search filter!")
        return items,errorCount,abs_path


    # Bulk Download
    def download(self,arguments):

        #for input coming from other python files
        if __name__ != "__main__":
            for arg in args_list:
                if arg not in arguments:
                    arguments[arg] = None

        ######Initialization and Validation of user arguments
        if arguments['keywords']:
            search_keyword = [str(item) for item in
arguments['keywords'].split(',')]

        if arguments['keywords_from_file']:
            search_keyword =
self.keywords_from_file(arguments['keywords_from_file'])

        # both time and time range should not be allowed in the same query
        if arguments['time'] and arguments['time_range']:
            raise ValueError('Either time or time range should be used in a
query. Both cannot be used at the same time.')

        # both time and time range should not be allowed in the same query
        if arguments['size'] and arguments['exact_size']:
            raise ValueError('Either "size" or "exact_size" should be used in
a query. Both cannot be used at the same time.')

        # both image directory and no image directory should not be allowed
in the same query
        if arguments['image_directory'] and arguments['no_directory']:
            raise ValueError('You can either specify image directory or
specify no image directory, not both!')

        # Additional words added to keywords
        if arguments['suffix_keywords']:
            suffix_keywords = [" " + str(sk) for sk in
```

```
arguments['suffix_keywords'].split(',')]
        else:
            suffix_keywords = ['']

        # Additional words added to keywords
        if arguments['prefix_keywords']:
            prefix_keywords = [str(sk) + " " for sk in
arguments['prefix_keywords'].split(',')]
        else:
            prefix_keywords = ['']

        # Setting limit on number of images to be downloaded
        if arguments['limit']:
            limit = int(arguments['limit'])
        else:
            limit = 100

        if arguments['url']:
            current_time = str(datetime.datetime.now()).split('.')[0]
            search_keyword = [current_time.replace(":", "_")]

        if arguments['similar_images']:
            current_time = str(datetime.datetime.now()).split('.')[0]
            search_keyword = [current_time.replace(":", "_")]

        # If single_image or url argument not present then keywords is
mandatory argument
        if arguments['single_image'] is None and arguments['url'] is None and
arguments['similar_images'] is None and \
                        arguments['keywords'] is None and
arguments['keywords_from_file'] is None:
            print('usage: google_images_download [-h] [-k KEYWORDS] [-sk
SUFFIX_KEYWORDS] [-pk PREFIX_KEYWORDS] [-l LIMIT] [-u URL] [-x SINGLE_IMAGE]\n'
                  'Uh oh! Keywords is a required argument \n\n'
                  'Please refer to the documentation on guide to writing
queries \n'

'https://github.com/hardikvasa/google-images-download#examples'
                  '\n\nexiting!\n'
                  '-------------------------------')
            sys.exit()


        # If this argument is present, set the custom output directory
        if arguments['output_directory']:
            main_directory = arguments['output_directory']
        else:
            main_directory = "downloads"

        # Proxy settings
        if arguments['proxy']:
            os.environ["http_proxy"] = arguments['proxy']
```

```
            os.environ["https_proxy"] = arguments['proxy']
            ######Initialization Complete

        paths = {}
        for pky in prefix_keywords:
            for sky in suffix_keywords:     # 1.for every suffix keywords
                i = 0
                while i < len(search_keyword):      # 2.for every main
keyword
                    iteration = "\n" + "Item no.: " + str(i + 1) + " -->" + "
Item name = " + str(pky) + str(search_keyword[i] + str(sky))
                    print(iteration)
                    print("Evaluating...")
                    search_term = pky + search_keyword[i] + sky

                    if arguments['image_directory']:
                        dir_name = arguments['image_directory']
                    elif arguments['no_directory']:
                        dir_name = ''
                    else:
                        dir_name = search_term + ('-' + arguments['color'] if
arguments['color'] else '')   #sub-directory


self.create_directories(main_directory,dir_name,arguments['thumbnail'])
#create directories in OS

                    params = self.build_url_parameters(arguments)
#building URL with params

                    url =
self.build_search_url(search_term,params,arguments['url'],arguments['similar_
images'],arguments['specific_site'],arguments['safe_search'])      #building
main search url

                    if limit < 101:
                        raw_html = self.download_page(url)  # download page
                    else:
                        raw_html =
self.download_extended_page(url,arguments['chromedriver'])

                    if arguments['no_download']:
                        print("Starting to Print Image URLS")
                    else:
                        print("Starting Download...")
                    items,errorCount,abs_path =
self._get_all_items(raw_html,main_directory,dir_name,limit,arguments)    #get
all image items and download images
                    paths[pky + search_keyword[i] + sky] = abs_path

                    #dumps into a json file
```

```
                    if arguments['extract_metadata']:
                        try:
                            if not os.path.exists("logs"):
                                os.makedirs("logs")
                        except OSError as e:
                            print(e)
                        json_file = open("logs/"+search_keyword[i]+".json",
"w")
                        json.dump(items, json_file, indent=4, sort_keys=True)
                        json_file.close()

                    #Related images
                    if arguments['related_images']:
                        print("\nGetting list of related keywords...this may
take a few moments")
                        tabs = self.get_all_tabs(raw_html)
                        for key, value in tabs.items():
                            final_search_term = (search_term + " - " + key)
                            print("\nNow Downloading - " + final_search_term)
                            if limit < 101:
                                new_raw_html = self.download_page(value)  #
download page
                            else:
                                new_raw_html =
self.download_extended_page(value,arguments['chromedriver'])
                            self.create_directories(main_directory,
final_search_term,arguments['thumbnail'])
                            self._get_all_items(new_raw_html, main_directory,
search_term + " - " + key, limit,arguments)

                    i += 1
                    print("\nErrors: " + str(errorCount) + "\n")
        if arguments['print_paths']:
            print(paths)
        return paths

#------------- Main Program -------------#
def main():
    records = user_input()
    for arguments in records:

        if arguments['single_image']:  # Download Single Image using a URL
            response = googleimagesdownload()
            response.single_image(arguments['single_image'])
        else:  # or download multiple images based on keywords/keyphrase
search
            t0 = time.time()  # start the timer
            response = googleimagesdownload()
            paths = response.download(arguments)  #wrapping response in a
variable just for consistency
```

```
            print("\nEverything downloaded!")
            t1 = time.time()  # stop the timer
            total_time = t1 - t0  # Calculating the total time required to
crawl, find and download all the links of 60,000 images
            print("Total time taken: " + str(total_time) + " Seconds")

if __name__ == "__main__":
    main()
```