Bandcamp Automator

December 1, 2022

[ ]:
```
from selenium.webdriver import Chrome
from selenium.webdriver.chrome.options import Options
opts = Options()
# opts.headless = True
browser = Chrome("./chromedriver.exe", options=opts)
browser.get('https://duckduckgo.com')
```

[ ]:
```
search_form = browser.find_element_by_id('search_form_input_homepage')
search_form.send_keys('real python')
search_form.submit()
```

[ ]:
```
results = browser.find_elements_by_class_name('results')
print(results[0].text)
```

[ ]:
```
browser.close()
```

[ ]:
```
opts = Options()
# opts.headless = True
browser = Chrome(options=opts)
browser.get('https://bandcamp.com')
browser.find_elements_by_class_name('playbutton')[0].click()
```

[ ]:
```
tracks = browser.find_elements_by_class_name('discover-item')
len(tracks)
tracks[3].click()
```

[ ]:
```
next_button = [e for e in browser.find_elements_by_class_name('item-page')
                   if e.text.lower().find('next') > -1]
next_button[0].click()
```

[ ]:
```
tracks = browser.find_elements_by_xpath("//div[@class='col col-3-12␣
 ↪discover-item']")[:8]
```

0.1 Building a Class

[1]:
```
from selenium.webdriver import Chrome
from selenium.webdriver.chrome.options import Options
from time import sleep, ctime
from collections import namedtuple
from threading import Thread
from os.path import isfile
import csv
```

[2]:
```
TrackRec = namedtuple('TrackRec', [
    'title',
    'artist',
    'artist_url',
    'album',
    'album_url',
    'timestamp'  # When you played it
])

BANDCAMP_FRONTPAGE='https://bandcamp.com/'
```

[3]:
```
class BandLeader():
    def __init__(self, headless=False, csv_path=None):
        # Create a headless browser
        opts = Options()
        opts.headless = headless
        self.browser = Chrome(options=opts)
        self.browser.get(BANDCAMP_FRONTPAGE)

        # Track list related state
        self._current_track_number = 1
        self.track_list = []
        self.tracks()

        # State for the database
        self.database = []
        if csv_path is not None:
            self.database_path = csv_path
        else:
            self.database_path = "./database.csv"

        if isfile(self.database_path):
            with open(self.database_path, newline='', encoding='utf-8') as␣
 ↪dbfile:
                dbreader = csv.reader(dbfile)
                next(dbreader)   # To ignore the header line
                self.database = [TrackRec._make(rec) for rec in dbreader]
        self._current_track_record = None

        # The database maintenance thread
        self.thread = Thread(target=self._maintain)
        self.thread.daemon = True     # Kills the thread with the main process␣
↪dies
        self.thread.start()

        self.tracks()


    def _maintain(self):
        while True:
            self._update_db()
            sleep(20)           # Check every 20 seconds


    def _update_db(self):
        try:
            check = (self._current_track_record is not None
                     and (len(self.database) == 0
                          or self.database[-1] != self._current_track_record)
                     and self.is_playing())
            if check:
                self.database.append(self._current_track_record)
                self.save_db()

        except Exception as e:
            print(f'error while updating the db: {e}')


    def tracks(self):

        '''
        Query the page to populate a list of available tracks.
        '''

        # Sleep to give the browser time to render and finish any animations
        sleep(2)

        # Get the container for the visible track list
        discover_section = self.browser.
↪find_element_by_class_name('discover-results')
        left_x = discover_section.location['x']
        right_x = left_x + discover_section.size['width']
        # Filter the items in the list to include only those we can click
        discover_items = self.browser.
↪find_elements_by_class_name('discover-item')
        self.track_list = [t for t in discover_items
                           if t.location['x'] >= left_x and t.location['x'] <␣
↪right_x]

        # Print the available tracks to the screen
#          for (i,track) in enumerate(self.track_list):
#              print('[{}]'.format(i+1))
#              lines = track.text.split('\n')
#              print(f'Album  : {lines[0]}')
#              print(f'Artist : {lines[1]}')
#              if len(lines) > 2:
#                  print(f'Genre  : {lines[2]}')


    def catalogue_pages(self):
        '''
        Print the available pages in the catalogue that are presently
        accessible.
        '''
        print('PAGES')
        for e in self.browser.find_elements_by_class_name('item-page'):
            print(e.text)
        print('')


    def more_tracks(self,page='next'):
        '''
        Advances the catalogue and repopulates the track list. We can pass in a␣
↪number
        to advance any of the available pages.
        '''

        next_btn = [e for e in self.browser.
↪find_elements_by_class_name('item-page')
                    if e.text.lower().strip() == str(page)]

        if next_btn:
            next_btn[0].click()
            self.tracks()

    def play(self,track=None):
        '''
        Play a track. If no track number is supplied, the presently selected␣
↪track
```

```
        will play.
        '''

        if track is None:
            self.browser.find_element_by_class_name('playbutton').click()
        elif type(track) is int and track <= len(self.track_list) and track >=␣
↪1:
            self._current_track_number = track
            self.track_list[self._current_track_number - 1].click()

        sleep(1)
        if self.is_playing():
            self._current_track_record = self.currently_playing()


    def play_next(self):
        '''
        Plays the next available track
        '''
        if self._current_track_number < len(self.track_list):
            self.play(self._current_track_number+1)
        else:
            self.more_tracks()
            self.play(1)


    def pause(self):
        '''
        Pauses the playback
        '''
        self.play()

    def is_playing(self):
        '''
        Returns `True` if a track is presently playing
        '''
        playbtn = self.browser.find_element_by_class_name('playbutton')
        return playbtn.get_attribute('class').find('playing') > -1


    def currently_playing(self):
        '''
        Returns the record for the currently playing track,
        or None if nothing is playing
        '''
        try:
            if self.is_playing():
                title = self.browser.find_element_by_class_name('title').text
                album_detail = self.browser.find_element_by_css_selector('.
↪detail-album > a')
                album_title = album_detail.text
                album_url = album_detail.get_attribute('href').split('?')[0]
                artist_detail = self.browser.find_element_by_css_selector('.
↪detail-artist > a')
                artist = artist_detail.text
                artist_url = artist_detail.get_attribute('href').split('?')[0]
                return TrackRec(title, artist, artist_url, album_title,␣
↪album_url, ctime())

        except Exception as e:
            print(f'there was an error: {e}')

        return None

    def save_db(self):
        with open(self.database_path,'w',newline='', encoding='utf-8') as␣
↪dbfile:
            dbwriter = csv.writer(dbfile)
            dbwriter.writerow(list(TrackRec._fields))
            for entry in self.database:
                dbwriter.writerow(list(entry))
```

[4]:
```
b = BandLeader()
```

[8]:
```
b.play()
```

[6]:
```
for i in range(10):
    b.play_next()
    sleep(3)
```

[9]:
```
b.pause()
b.save_db()
b.browser.close()
```

0.2 Credits

Modern Web Automation With Python and Selenium