```python
# fs sampler
# version: 0.1
# Dieses Programm dient dazu anhand von spezifischen Parametern aus der online Library Free Sound Dateien zu selektieren und diese in ein Verzeichnis lokal runter zu laden

from __future__ import print_function
import freesound
import os
import sys
from oauthlib.oauth2 import BackendApplicationClient
from requests_oauthlib import OAuth2Session
import hashlib
import random
import time
import subprocess

# OSC
from osc4py3.as_eventloop import *
from osc4py3 import oscmethod as osm

class config():
    """
    Start here to edit your config of the program
    """
    APIKEY="Ld15pPetuu7VMVOGEzgkrvTp23IaiplOUmuszK44"
    OAUTHTOKEN="VegqtgKdHqbR0KwcElFha5FvC2vhhQ"
    FETCHDIRNAME = "fetchedSounds"
    COUNT = 10 #Maximale Anzahl der Sounds (-1 fÃ¼r unbegrenzt)
    PAGESIZE = 300 #Ergebnisse pro Abruf von Freesounds INT (Seitenbasiert, Seite n kann spezifiziert werden)
    DEBUG = True
    MINSOUNDDURATION = 1
    MAXSOUNDDURATION = 60

class fsFetcher():
    def __init__(self):
        self.fsClient = freesound.FreesoundClient()
        self.fsClient.set_token(config.APIKEY)
        #programm status (True = weiter arbeiten, False = Fehler)
        self.state = False

    def createDirs(self):
        self.path_name = os.path.join(os.getcwd(), config.FETCHDIRNAME)
        if config.DEBUG:
            print ("directory path:")
            print(self.path_name)
        try:
            if config.DEBUG:
                print("creating dir for previews...")
            os.mkdir(self.path_name)
        except (FileExistsError):
            if config.DEBUG:
                print ("dir already created, skipping")
        except:
            if config.DEBUG:
                print ("cannot create folder: "+self.path_name+" ! Cannot continue")
            return False
        try:
            if config.DEBUG:
                print ("creating dir for wave files...")
            os.mkdir(self.path_name+'/wav')
        except (FileExistsError):
            if config.DEBUG:
                print ("dir already created, skipping")
        except:
            if config.DEBUG:
                print ("cannot create folder: "+self.path_name+"/wav ! Cannot continue")
            return False
        self.state = True
        return True

    def md5(self,fname):
        hash_md5 = hashlib.md5()
        with open(fname, "rb") as f:
            for chunk in iter(lambda: f.read(4096), b""):
                hash_md5.update(chunk)
        return hash_md5.hexdigest()

    def selectSounds(self, minDuration=config.MINSOUNDDURATION, maxDuration=config.MAXSOUNDDURATION, geo=[-10,52,4000]):
        """
        Diese Methode wÃ¤hlt Sounds aufgrund folgender Parameter aus der Freesound Library
        Parameter: minimale Dauer, maximale Dauer, Geotags: Breitengrad, LÃ¤ngengrad, Entfernung (Radius) als INT

        Text Search Request:
       >>> sounds = c.text_search(
       >>>     query="dubstep", filter="tag:loop", fields="id,name,url"
       >>> )
       >>> for snd in sounds: print snd.name

        Geotag Filter:
            #filter={!geofilt sfield=geotag pt=<LATITUDE>,<LONGITUDE> d=<MAX_DISTANCE_IN_KM>}
        """
        if not (self.state):
            return False
        if config.DEBUG:
            print ("fetching sound data from freesounds")
            print ("geotags:")
            print (geo)
        soundGeoTagging = geo
        start = time.time()
        queryFilter = "{{!geofilt sfield=geotag pt={0},{1} d={2}}}".format(geo[0],geo[1],geo[2])
        #queryFilter = "type:wav {!geofilt sfield=geotag pt=13,52 d=2000}"
        queryFields ="id,name,duration,md5,type,previews"
        sounds = self.fsClient.text_search(filter=queryFilter,fields=queryFields, page_size=config.PAGESIZE)
        stop  = time.time()
        if config.DEBUG:
            print ("dauer fÃ¼r freesounds abruf: ")
            print (stop-start)
        return self.filterByDuration(sounds, minDuration, maxDuration)

    def downloadSounds(self, soundsObject):
        if not (self.state):
            return False
        #self.fsClient.set_token(config.OAUTHTOKEN, "oauth")
        """
        Download Sound Files
        Erwartet ein Sound Objekt mit einer Liste von Sounds
        """
        i = 0
        for sound in soundsObject:
            if (i >= 0) & (i < config.COUNT):
                self.nameFileByIndex(sound, i)
                #filename = "sound_"+str(i)+".wav"
                                #sound.retrieve_preview(self.path_name, name=filename)
                i += 1
            else:
                return
        return
    def nameFileByIndex(self, soundObject, i):
        if not (self.state):
            return False
        filename = "sound_"+str(i)
        if config.DEBUG:
            print("\t\tDownloading:", soundObject.name)
            print("as: "+filename)
        soundObject.retrieve_preview(self.path_name, name=filename)
        """
        Loglevel ffmpeg
            -loglevel [repeat+]loglevel | -v [repeat+]loglevel
            Set the logging level used by the library.
            â€˜quiet, -8â€™
            Show nothing at all; be silent.
            â€˜panic, 0â€™
            Only show fatal errors which could lead the process to crash, such as an assertion failure. This is not currently used for anything.
            â€˜fatal, 8â€™
            Only show fatal errors. These are errors after which the process absolutely cannot continue.
            â€˜error, 16â€™
            Show all errors, including ones which can be recovered from.
            â€˜warning, 24â€™
            Show all warnings and errors. Any message related to possibly incorrect or unexpected events will be shown.
            â€˜info, 32â€™
            Show informative messages during processing. This is in addition to warnings and errors. This is the default value.
            â€˜verbose, 40â€™
            Same as info, except more verbose.
            â€˜debug, 48â€™
            Show everything, including debugging information.
            â€˜trace, 56â€™
        """
        subprocess.call(['ffmpeg','-v', 'warning', '-y', '-i', self.path_name+'/'+filename,self.path_name+'/wav/'+filename+'.wav'])
        return

    def nameFileByName(self, soundObject):
        if not (self.state):
            return False
        fullfilepath = self.path_name+"/"+soundObject.name
        if (os.path.isfile(fullfilepath)):
            if config.DEBUG:
                print ("dateiname vorhanden: "+soundObject.name)
        else:
            if config.DEBUG:
                print ("datei muss geladen werden:")
                print("\t\tDownloading:", soundObject.name)
            #if sound.name.endswith(sound.type):
            filename = soundObject.name
            soundObject.retrieve_preview(self.path_name, name=filename)
            #else:
            #   filename = "%s.%s" % (sound.name, sound.type)
            #   sound.retrieve_preview(self.path_name, name=filename)
        return

    def filterByDuration(self, soundsObject, minDuration, maxDuration):
        if not (self.state):
            return False
        sounds = soundsObject
        soundList = []
        tmp = time.time()
        for sound in sounds:
            soundList += [sound]
        random.shuffle(soundList)
        filteredObjects = []
        i = 0
        if config.DEBUG:
            print ("dauer fÃ¼r shuffle: ")
            print (time.time()-tmp)
            print ("Preselected Sounds:")
            print (soundList)
        for sound in soundList:
            if (i >= 0) & (i < config.COUNT):
                if (int(sound.duration) >= minDuration) & (int(sound.duration) <= maxDuration):
                    filteredObjects += [sound]
                    i += 1
        if config.DEBUG:
            print ("selected sounds after filtering:")
            print (filteredObjects)
        return filteredObjects

```

```python
class OSCListener():
    def __init__(self):
        self.serverip = "127.0.0.1"
        self.port = 12000
        #Programaufruf
        self.appStart = time.time()
        self.soundFetcher = fsFetcher()
        self.soundFetcher.createDirs()
    def handlerForWLR(self, x, y, z):
        # Will receive message data unpacked in x,yz
        # Koordignatenaufruf, Download
        sounds = self.soundFetcher.selectSounds(geo=[x,y,z])
        download = self.soundFetcher.downloadSounds(sounds)


    def startup(self):
        # Start the system.
        if config.DEBUG:
            print("starting up Server...")
        osc_startup()

        # Make server channels to receive packets.
        osc_udp_server(self.serverip, self.port, "aservername")

        # Associate Python functions with message address patterns, using default
        # argument scheme OSCARG_DATAUNPACK.
        osc_method("/incommingWLR*", self.handlerForWLR)
        if config.DEBUG:
            print("listening...")

    def shutdown(self):
        osc_terminate()

    def listenLoop(self):
        # Periodically call osc4py3 processing method in your event loop.
        finished = False
        while not finished:
            try:
                osc_process()
            except KeyboardInterrupt:
                finished = True
                osc_terminate()
                raise

        # Properly close the system.
        self.shutdown()


server = OSCListener()
server.startup()
server.listenLoop()

appStop  = time.time()
print ("Programmdauer: ")
print  (appStop - appStart)
```