```
from __future__ import unicode_literals
import hashlib
from argparse import ArgumentParser, ArgumentDefaultsHelpFormatter
from os import mkdir
from os.path import basename, splitext, join, isfile, isdir, exists
from xml.etree.cElementTree import Element, SubElement, ElementTree

import os, sys


class Main:
    def __init__(self):
        self.options = None

    def main(self):
        """Developed for the LTP project, this tool scans the folder SRC. Each detected
file is copied to the folder
         DST and an additional sidecar is placed alongside it, with the same name but with
extension XML. To prevent
         accidentally overwriting files the DST folder must be empty at start. The folder
SRC and DST can be in separate
         partitions."""
        parser = ArgumentParser(description=Main.main.__doc__,
formatter_class=ArgumentDefaultsHelpFormatter)
        parser.add_argument(dest='SRC', type=unicode, default=None, help='Source folder to
process')
        parser.add_argument(dest='DST', type=unicode, default=None, help='Destination
folder where to copy the file and place the sidecar')
        parser.add_argument(dest='LOT', type=unicode, default=None, help='Name of the
lot')
        parser.add_argument('-R', dest='RECURSIVE', action='store_true', default=False,
help='If true, also process the subfolders')
        parser.add_argument('-E', dest='EXTENSION', action='store_true', default=False,
help='If true, also include the extension of the original file in the file name of the
copy, e.g. myfile.pdf -> myfile_pdf.pdf')
        self.options = parser.parse_args()

        # Validate src is folder
        src_folder = self.options.SRC
        if not exists(src_folder):
            self.exit('SRC does not exist: {}'.format(src_folder), -1)
        if not isdir(src_folder):
            self.exit('SRC is not a folder: {}'.format(src_folder), -1)

        # Validate dst folder
        dst_folder = self.options.DST
        if not exists(dst_folder):
            self.exit('DST does not exist: {}'.format(dst_folder), -1)
        if not isdir(dst_folder):
            self.exit('DST is not a folder: {}'.format(dst_folder), -1)
        if self.get_folders(dst_folder) or self.get_files(dst_folder):
            self.exit('DST folder is not empty. Please provide an empty folder.:
{}'.format(dst_folder), -1)

        # Proceed
        self.process_folder(src_folder, dst_folder)
        self.exit('Successfully processed all folders')

    def process_folder(self, src_folder, dst_folder):
        print('Processing folder: {} -> {}'.format(src_folder, dst_folder))

        # Process the files
        processed = []
        files = self.get_files(src_folder)
        for index, src_file in enumerate(files):
            # Skip XML files
```

```
            base, ext = splitext(src_file)
            if ext.lower() == '.xml':
                print('WARNING: Skipping XML file: {}'.format(src_file))
                continue

            # Already processed?
            if not self.options.EXTENSION:
                if base in processed:
                    print('WARNING: File with same root (filename without extension)
already has been processed in this folder: {}'.format(src_file))
                    continue
            processed.append(base)

            # Process
            print('Processing file {}/{}: {}'.format(index + 1, len(files), src_file))
            dst_file = join(dst_folder, basename(src_file))
            if self.options.EXTENSION:
                root, ext = splitext(dst_file)
                if ext.startswith('.'):
                    ext = ext[1:]
                dst_file = '{}_{}.{}'.format(root, ext, ext)

            self.process_file(src_file, dst_file)

        # Recurse?
        if not self.options.RECURSIVE:
            return

        # Process the subfolders
        for src_subfolder in self.get_folders(src_folder):
            dst_subfolder = join(dst_folder, basename(src_subfolder))
            mkdir(dst_subfolder)
            self.process_folder(src_subfolder, dst_subfolder)

    def process_file(self, src, dst):
        md5 = self.copy_file(src, dst)
        self.copy_sidecar(md5, dst)

    def copy_file(self, src, dst):
        buffer = memoryview(bytearray(2 ** 16))
        hash = hashlib.md5()

        with open(src, 'rb') as fp_src:
            with open(dst, 'wb') as fp_dst:
                bytes = fp_src.readinto(buffer)
                while bytes > 0:
                    hash.update(buffer[:bytes])
                    fp_dst.write(buffer[:bytes])
                    bytes = fp_src.readinto(buffer)

        return hash.hexdigest().upper()

    def copy_sidecar(self, md5, dst):
        # Create the sidecar XML
        title, _ = splitext(basename(dst))
        #current_datetime = datetime.now()

        #code = "lot-" + current_datetime.strftime("%m_%d_%Y, %H_%M_%S")
        code = self.options.LOT
        print(code)
        #/kbin-irsnb/out_python/BE-RBINS-INV-MT-145-Byssanodonta-ovata/BE-RBINS-INV-
MT-145-Byssanodonta-ovata-25-09-2023-14h39-PH31.zip.zip
        pathfilearray = dst.split("/")
        pathfile = pathfilearray[7]
        root = Element('sip')
        root.set('version', '2.0')
```

```
        archive_element = SubElement(root, 'archive')
        code_element = SubElement(archive_element, 'code')
        code_element.text = code
        label_element = SubElement(archive_element, 'label')
        label_element.text = title
        documents_element = SubElement(archive_element, 'documents')
        document_element = SubElement(documents_element, 'document')
        file_element = SubElement(document_element, 'file')
        path_element = SubElement(file_element, 'path')
        path_element.text = pathfile
        md5_element = SubElement(file_element, 'checksum')
        md5_element.set('type', 'MD5')
        md5_element.text = md5
        tree = ElementTree(root)

        # Save at destination
        sidecar = self.change_ext(dst, 'xml')
        tree.write(sidecar)

    def exit(self, msg, status=0):
        print(msg)
        sys.exit(status)

    @staticmethod
    def change_ext(file, ext):
        root, old = splitext(file)
        return root + '.' + ext

    @staticmethod
    def get_files(folder):
        return [join(folder, item) for item in os.listdir(folder) if isfile(join(folder,
item))]

    @staticmethod
    def get_folders(folder):
        return [join(folder, item) for item in os.listdir(folder) if isdir(join(folder,
item))]


if __name__ == '__main__':
    Main().main()
```