Assignment 13

Python programming

LILLEBAELT ACADEMY OF
PROFESSIONAL HIGHER EDUCATION

Author
Martin Grønholdt
mart80c7@edu.eal.dk

Wednesday 3 May 2017

Table of Contents

1. Introduction

This document describes a Python program to count all lines of python code in a given path.

All hand ins for this course is available on GitHub at:
https://github.com/deadbok/eal_programming

2. Running

The goal of this program is to count the number of lines in Python files. It does so recursively starting at the directories given on the command line. The program uses the argparse Python package and includes command line help:

```
$ python3 countcode.py --help
codecount.py v1.5.0 by Martin B. K. Grønholdt

usage: countcode.py [-h] [--exclude [EXCLUDE [EXCLUDE ...]]]
                   include [include ...]

Count lines of Python code in given paths.

positional arguments:
 include               Path to include

optional arguments:
 -h, --help            show this help message and exit
 --exclude [EXCLUDE [EXCLUDE ...]]
                       Path to exclude
```

The command line help message.

The above states that the program expect a list of directories to scan, having at least one entry. "--exclude" makes it possible to exclude certain path from the scanning by listing them after the parameter. Excluded directories match the first part of the path, any path that start with a path in the exclude list, are excluded.

```
$ python3 countcode.py . --exclude ./tools
codecount.py v1.5.0 by Martin B. K. Grønholdt

Include paths:
       .
Exclude paths:
       ./tools

Scanning selected Python files...
c       ./countcode.py, b:56,c:41,p:184,t281
e       ./tools/py2puml.py, ./tools

Totals:
       Files: 2
       Duplicate files: 0
       Excluded files: 1
       Lines: 281
       Blank lines: 56
       Comment lines: 41
       Python code lines: 184
```

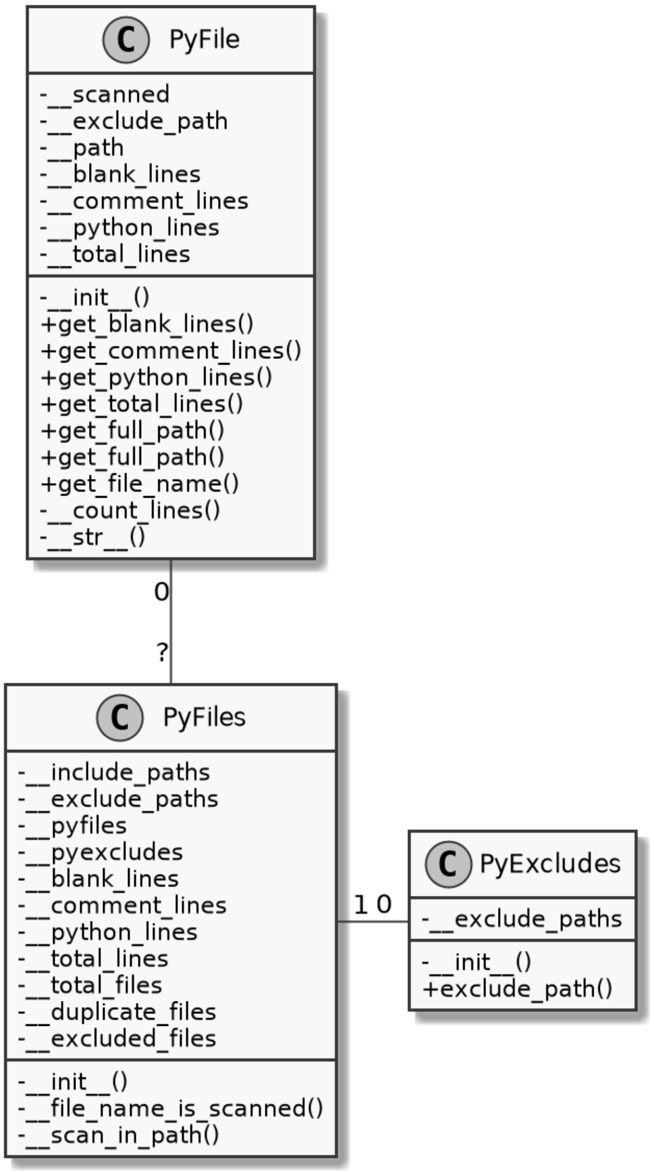
Example of running the code in the project root directory.

The example above asks the program to count all Python file lines in the current directory (.) but exclude files in the tools directory (--exclude ./tools).

- The scanning output starts with a status character:
  - "c" – The file contents are included in the count.
  - "e" - The file is excluded from the count.
  - "d" - The file is excluded from the count, because it is an exact duplicate of another file.
- Next comes the file name and path of the file.
- If the file was included in the count the counts are listed:
  - "b:" - Followed by the number of blank lines.
  - "c:" - Followed by the number of comment lines.
  - "p:" - Followed by the number of Python code lines.
  - "t:" - Followed by the number of total lines.
- If the file is excluded or a duplicate, the offending path is listed instead.

3. Classes

- PyFiles – The main class that takes care of recursing the directories and finding the Python source files, it uses os.walk and fnmatch.filter() to accomplish this. It uses an instance of PyExclude to check if the file is in one of the path that are to be excluded. It uses a list of instances of PyFile to keep the file data, and prints the total count when done.
- PyFile – This class keeps a the relevant data for, and counts the source lines of each file. It uses a very simple algorithm to try and distinguish between blank, comment, and source lines and also formats the file status output message.
- PyExclude – This is a helper class used to tell if a file is in the excluded paths.



Class diagram

4. countcode.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
"""
Name: countcode.py
Author: Martin Bo Kristensen Grønholdt.
Version: 1.5.0 (2017-05-03)

Count the lines of Python code in files in the paths given on the commend line.
"""
import argparse
import os
import fnmatch
import filecmp

# Program version.
__VERSION__ = '1.5.0'


class PyExcludes:
    """
    Helper class to tell if a certain path is to be excluded from the scan.
    """

    def __init__(self, exclude_paths=[]):
        """
        Constructor.

        :param exclude_paths: List of exclude paths.
        """
        self.__exclude_paths = exclude_paths

    def exclude_path(self, path):
        """
        Return relevant exclude path or None.

        :param path: Path to match against the exclude paths.
        :return: Matching exclude path or None.
        """
        # Could have just used return(ex), but my teacher would rather like
        # only one return path.
        ret = None
        for ex in self.__exclude_paths:
            # If path is in the exclude paths stop and return it.
            if path.startswith(ex):
                ret = ex
                break

        return ret


class PyFile:
    """
    Class for scnning and managing info about a single Python file.
    """
    def __init__(self, path=None, exclude_path=None, duplicate_path=None):
        """
        Constructor.

        :param path: Path of the Python file.
        :param exclude_path: Relevant exclude path or None.
        :param duplicate_path: Relevant duplicate path or None.
        """
        # File has not been scanned.
        self.__scanned = False
        # Save exclude path.
        self.__exclude_path = exclude_path

        if exclude_path is not None:
            # File is excluded, set status and do not scan.
            self.__status = 'e'
            self.__scanned = True
        elif duplicate_path is not None:
            # File is a duplicate, set status and do not scan.
            self.__status = 'd'
            self.__scanned = True
            self.__exclude_path = duplicate_path
        else:
            # File is to be counted, set status and scan.
            self.__status = 'c'

        # Initialise variables for counting.
        self.__path = path
        self.__blank_lines = 0
        self.__comment_lines = 0
        self.__python_lines = 0
        self.__total_lines = 0

        # Count the code lines.
        self.__count_lines()

    def get_blank_lines(self):
        """
        Return the number of blank lines in the file.
        """
        return self.__blank_lines

    def get_comment_lines(self):
        """
        Return the number of comment lines in the file.
        """
        return self.__comment_lines

    def get_python_lines(self):
        """
        Return the number of python source lines in the file.
        """
        return self.__python_lines

    def get_total_lines(self):
        """
        Return the total number of lines in the file.
        """
        return self.__total_lines

    def get_full_path(self):
        """
        Return the full path of the file.
        """
        return self.__path

    def get_file_name(self):
        """
        Return the file name of the file.
        """
        return os.path.basename(self.__path)

    def __count_lines(self):
        """
        Count the lines of a Python file.

        :param path: Path to file to process.
        """
        # Check if the file is excluded or scanned.
        if (self.__exclude_path is None) and (not self.__scanned):
            # Open the file.
            with open(self.__path, 'r', encoding="latin-1") as python_file:
                # Reset counters.
                self.__blank_lines = 0
                self.__comment_lines = 0
                self.__python_lines = 0
                self.__total_lines = 0

                # Run through all lines in the file.
                for line in python_file.readlines():
                    # Add to total lines.
                    self.__total_lines += 1

                    # Remove trailing spaces and tabs.
                    line = line.lstrip(' \t')

                    # Check for a blank line.
                    if line.startswith('\n'):
                        self.__blank_lines += 1
                    # Check for a comment.
                    elif line.startswith('#'):
                        self.__comment_lines += 1
                    # Count as a line of Python code.
                    else:
                        self.__python_lines += 1

            # File is scanned.
            self.__scanned = True

    def __str__(self):
        """
        Return a nice status line for the file.
        :return: Status string.
        """
        ret = '{0}\t{1}'.format(self.__status,
                                self.__path)
        if (self.__exclude_path is None):
            ret += ', b:{0},c:{1},p:{2},t{3}'.format(self.__blank_lines,
                                                      self.__comment_lines,
                                                      self.__python_lines,
                                                      self.__total_lines)

        if (self.__exclude_path is not None):
            ret += ', {0}'.format(self.__exclude_path)

        return ret


class PyFiles:
    """
    Class for recursively scanning path and managing file info.
    """

    def __init__(self, include_paths=['.'], exclude_paths=None):
        """
        Constructor.

        :param include_paths: List of path to include when scanning (defaults
                              to '.').
        :param exclude_paths: List of paths to exclude when scanning
                              (defaults to None).
        """
        # Save paths
        self.__include_paths = include_paths
        self.__exclude_paths = exclude_paths

        # Array of PyFiles to keep the file information.
        self.__pyfiles = []

        # Excluded paths handler.
        self.__pyexcludes = PyExcludes(exclude_paths)

        # Print paths
        print('Include paths:')
        for path in self.__include_paths:
            print('\t{}'.format(path))
        if self.__exclude_paths is not None:
            print('Exclude paths:')
            for path in self.__exclude_paths:
                print('\t{}'.format(path))

        # Initialise variables used during the counting.
        self.__blank_lines = 0
        self.__comment_lines = 0
        self.__python_lines = 0
        self.__total_lines = 0

        self.__total_files = 0
        self.__duplicate_files = 0
        self.__excluded_files = 0

        # Scan all files.
        print('\nScanning selected Python files...')
        for path in self.__include_paths:
            self.__scan_in_path(path)
        # Sum the number of lines.
        for pyfile in self.__pyfiles:
            self.__blank_lines += pyfile.get_blank_lines()
            self.__comment_lines += pyfile.get_comment_lines()
            self.__python_lines += pyfile.get_python_lines()
            self.__total_lines += pyfile.get_total_lines()

        # Print result.
        print('\nTotals:')
        print('\tFiles: {}'.format(self.__total_files))
        print('\tDuplicate files: {}'.format(self.__duplicate_files))
        print('\tExcluded files: {}'.format(self.__excluded_files))
        print('\tLines: {}'.format(self.__total_lines))
        print('\tBlank lines: {}'.format(self.__blank_lines))
        print('\tComment lines: {}'.format(self.__comment_lines))
        print('\tPython code lines: {}'.format(self.__python_lines))

    def __file_name_is_scanned(self, filename):
        """
        Check if a file name has been seen before.

        :param filename: The file name to check.
        :return: Return the full path og the matching file name or None.
        """
        # Could have just used return(pyfile.get_full_path()), but my teacher
        # would rather like only one return path.
        ret = None
        for pyfile in self.__pyfiles:
            # Get out if the file name is known.
            if pyfile.get_file_name() == filename:
                ret = pyfile.get_full_path()
                break

        return ret

    def __scan_in_path(self, path):
        """
        Scan a path for Python files and count the lines of the relevant ones.

        :param path: Path to scan.
        """
        # Use os.walk to recursively decent the directory.
        for root, dirnames, filenames in os.walk(path):
            # Find all .py files in the path.
            for filename in fnmatch.filter(filenames, '*.py'):
                # Assemble the current path.
                current_path = os.path.join(root, filename)

                # One more file.
                self.__total_files += 1

                # Get relevant exclude path if any.
                exclude_path = self.__pyexcludes.exclude_path(current_path)

                # Check for duplicates if not excluded.
                duplicate_path = None
                if exclude_path is None:
                    # Check for duplicate file name.
                    duplicate_path = self.__file_name_is_scanned(filename)
                    if duplicate_path is not None:
                        # Compare the files.
                        if not filecmp.cmp(duplicate_path, current_path):
                            # It is not a duplicate.
                            duplicate_path = None
                        else:
                            # It is a duplicate, up the count.
                            self.__duplicate_files += 1
                else:
                    # The file is ecluded, up the count.
                    self.__excluded_files += 1

                # Create the file record, and scan it if needed.
                pyfile = PyFile(current_path,
                                exclude_path,
                                duplicate_path)
                self.__pyfiles.append(pyfile)
                # Print a status line for the file.
                print(pyfile)


def parse_commandline():
    """
    Parse command line arguments.

    :return: A tuple with a list of files path to include, and a list of paths
             to exclude.
    """
    # Set up the arguments to have included paths as positional arguments,
    # and excluded path using --exclude.
    parser = argparse.ArgumentParser(description='Count lines of Python ' +
                                                 'code in given paths.')
    # Include parameter, at least one entry is needed.
    parser.add_argument('include', default=['.'], nargs='+',
                        help='Path to include')
    # Eclude paramater.
    parser.add_argument('--exclude', dest='exclude', nargs='*',
                        help='Path to exclude')

    # Parse command line
    args = parser.parse_args()

    # Return the paths.
    return ((args.include, args.exclude))


def main():
    """
    Program main entry point.
    """
    # Print welcome message
    print(
        'codecount.py v{} by Martin B. K. Grønholdt\n'.format(__VERSION__))
    # Parse the command line.
    paths = parse_commandline()

    # Start counting.
    PyFiles(paths[0], paths[1])
```

```
# Run this when invoked directly
if __name__ == '__main__':
    main()
```

5. Result

```
$ python3 countcode.py ../.. /____/_______/sync-src /____/_______/src --exclude /____/_______/sync-src/.metadata \
/____/_______/sync-src/python/ssg/build /____/_______/sync-src/enigmabox-openwrt /____/_______/sync-src/lcdproc \
/____/_______/sync-src/cg/commgroup.dk /____/_______/sync-src/Static\ Site\ Generator/ \
/____/_______/src/.metadata /____/_______/src/python/ssg/build /____/_______/src/enigmabox-openwrt \
/____/_______/src/lcdproc \ /____/_______/src/cg/commgroup.dk /____/_______/src/Static\ Site\ Generator/
codecount.py v1.5.0 by Martin B. K. Grønholdt

Include paths:
       ../..
       /____/_______/sync-src
       /____/_______/src
Exclude paths:
       /____/_______/sync-src/.metadata
       /____/_______/sync-src/python/ssg/build
       /____/_______/sync-src/enigmabox-openwrt
       /____/_______/sync-src/lcdproc
       /____/_______/sync-src/cg/commgroup.dk
       /____/_______/sync-src/Static Site Generator/
       /____/_______/src/.metadata
       /____/_______/src/python/ssg/build
       /____/_______/src/enigmabox-openwrt
       /____/_______/src/lcdproc
       /____/_______/src/cg/commgroup.dk
       /____/_______/src/Static Site Generator/

Scanning selected Python files...
---
c       ../../programming/ass13/countcode.py, b:63,c:60,p:234,t357
c       ../../programming/ass13/tools/py2puml.py, b:28,c:64,p:101,t193
c       ../../programming/ass14/customer.py, b:15,c:7,p:72,t94
c       ../../programming/ass14/employee.py, b:13,c:6,p:46,t65
---
c       ../../programming/SQLite 10/config.py, b:1,c:3,p:7,t11
c       ../../programming/SQLite 10/run.py, b:1,c:3,p:7,t11
d       ../../programming/SQLite 10/__init__.py, ../../intro-project/roy/motor/ws/__init__.py
c       ../../programming/SQLite 10/app/__init__.py, b:3,c:4,p:26,t33
```

```
c       ../../programming/SQLite 10/app/models/customerdb.py, b:13,c:17,p:82,t112
c       ../../programming/SQLite 10/app/models/__init__.py, b:0,c:0,p:1,t1
c       ../../programming/SQLite 10/app/views/add.py, b:6,c:4,p:28,t38
c       ../../programming/SQLite 10/app/views/customertable.py, b:4,c:3,p:18,t25
c       ../../programming/SQLite 10/app/views/__init__.py, b:0,c:0,p:2,t2
---
e       /____/_______/sync-src/lcdproc/setup.py, /____/_______/sync-src/lcdproc
e       /____/_______/sync-src/lcdproc/examples.py, /____/_______/sync-src/lcdproc
e       /____/_______/sync-src/lcdproc/ez_setup.py, /____/_______/sync-src/lcdproc
e       /____/_______/sync-src/lcdproc/lcdproc/__init__.py, /____/_______/sync-src/lcdproc
e       /____/_______/sync-src/lcdproc/lcdproc/screen.py, /____/_______/sync-src/lcdproc
e       /____/_______/sync-src/lcdproc/lcdproc/widgets.py, /____/_______/sync-src/lcdproc
e       /____/_______/sync-src/lcdproc/lcdproc/server.py, /____/_______/sync-src/lcdproc
e       /____/_______/sync-src/lcdproc/lcdproc/menu.py, /____/_______/sync-src/lcdproc
---

Totals:
       Files: 860
       Duplicate files: 249
       Excluded files: 142
       Lines: 42445
       Blank lines: 6760
       Comment lines: 5041
       Python code lines: 30644
```

Program output, it has been truncated at the "---" markers and certain directories have had their characters replaced by underscores.