Assignment: Scripting in Python

Introduction

The lecturer of the course of Advanced Programming is experiencing some troubles while correcting the solutions proposed by the students to the programming assignments published along the course. Therefore as the last programming assignment, the students are asked to write some Python scripts that will help the lecturer in the evaluation.
Expected output: A single Python file called `ap18helper.py` containing all the functions described in the following exercises.
The Python code should adhere to the following guidelines:

1. **Style**: The code should be written adhering to the PEP 8 – Style Guide for Python Code
2. **Documentation**: The code has to be adequately commented. Every function must have a `docstring` according to PEP 257
3. The code has to be written in a well-structured way, avoiding redundancies and factoring out common functionalities in auxiliary functions.
4. Every function should inform the user of any action that modifies the file system. Rather than using `print`, the scripts should use the module `logging` of the standard library, recording with `logging.info()` all such actions. The logging level has to be set to `logging.INFO` (for example with `logging.basicConfig(level=logging.INFO)`).
5. **Exceptions**: No sophisticated exception handling is requested, but the Python code should never crash abruptly because of an exception. If an unexpected exception is raised, the script should exit gracefully after printing an error message.

Exercise 1 - Renaming archives (`raj2jar`)

Since some mail servers do not allow `.jar` files as attachments, the lecturer asked the students to rename all such archives by changing the extension from `.jar` to `.raj`. Now, to test the submitted JavaBeans the inverse renaming has to be performed.
Write a Python function called `raj2jar` that takes a string `root` as parameter, which is the absolute path of a directory in the local file system. The script must rename all files with extension `.raj` in the subtree of directories rooted at `root` by changing the extension to `.jar`.

Exercise 2 - Producing the list of files to be printed (`collect_sources`)

To correct the solutions submitted by a student, the lecturer needs to print all the source files submitted by her/him. To do this, he needs a file containing the paths of all Java, Haskell and Python source files produced by the student.
Write a Python function called `collect_sources` that takes as parameter two strings: `root`, which is an absolute path, and `sources`, which is the name of a text file. The script must collect in the file `sources` placed in directory `root` the relative paths (using Linux separators) starting from `root` of all files with extension `.java`, `.hs` or `.py` contained in the subtree of directories rooted at `root`.

For example, if `root` is `"C:\Users\Fake"` and directory `C:\Users\Fake` has the following structure

```
Fake\dir1\file1.java
         \file2.hs
         \arch1.zip
    \dir2\dir3\Winner.java
              \WinnerDB.java
              \oscar-winners.csv
         \dir4\Ex1.hs
              \Ex3.hs
    \dir3\MyBeautifulScript.py
         \MyBeautifulScript.pyc
```

then as a result of calling `collect_sources("C:\Users\Fake","src.txt")` the final contents of file `C:\Users\Fake\src.txt` should be

```
dir1/file1.java
dir1/file2.hs
dir2/dir3/Winner.java
dir2/dir3/WinnerDB.java
dir2/dir4/Ex1.hs
dir2/dir4/Ex3.hs
dir3/MyBeautifulScript.py
```

Exercise 3 - Rebuilding the structure of packages (`rebuild_packages`)

When starting a project in Java, some IDE gently persuades the programmer to place the classes in a new package, which in most situations is a new directory having the same name as the package. However, the lecturer asked the students to send only the Java files as solutions, not the whole tree structure. This causes some problems when compiling or executing Java files, if they are not in a directory named as the declared package.
Write a Python function `rebuild_packages` that takes a string `root` as a parameter, which is an absolute path. For each Java file in the subtree rooted at `root`, the script must ensure that if the file includes a `package` statement, then it is in a directory with the same name of the package. More precisely:

1. if the Java file does not contain a `package` statement, nothing to do;
2. if the Java file contains a `package pname;` statement (thus `pname` is the package to which the class belongs), and the file is in a directory called `pname`, nothing to do;
3. if the Java file contains a `package pname;` statement, the name of the directory of the file is different from `pname` and there is a subdirectory called `pname`, then the Java file has to be moved to that subdirectory;
4. finally, if the Java file contains a `package pname;` statement, the name of the directory of the file is different from `pname` and no subdirectory called `pname` exists, then the script must create a subdirectory called `pname` and the Java file has to be moved there.

Exercise 4 - Retrieving the files for testing (`download_tests`)

The lecturer and the tutor prepared (or will prepare...) some files for testing the solutions submitted by the students. However, such files have to be copied in the right directories, once for each student, and this is a tedious and repetitive task.
The list of files and additional information needed to test each of the submitted solutions is recorded in the CSV (comma-separated value) file `AP_TestRegistry.csv`, having the following fields

```
filename,    testfiles,    command
```

where `filename` is the name of the file to be tested, including the extension (that you can assume to be in `{".java", ".hs", ".py"}`, `testfiles` is a colon-separated (:) list of test file names (possibly empty), and `command` is a string to be executed in a shell to test `filename`.
Since the registry and the test files are changed frequently, before executing the tests one has to download their latest versions from the URL `http://pages.di.unipi.it/corradini/Didattica/AP-18/PROG-ASS/03/Test`
Write a function `download_tests` that takes as argument a string `root`, which is an absolute path in the local file system, and a string representing a `URL`. The second argument must be optional and must have the URL written above as default. The function must do the following:

1. download the file `AP_TestRegistry.csv` from `URL`
2. for each entry in the registry, if there is a file called `filename` in the subtree of directories rooted at `root`, then copy from `URL` all the files in `testfiles` in the **testing directory** of `filename`. Note that
   - if a file with the same name exists in the testing directory, it has to be overwritten;
   - the *testing directory* of a Haskell or Python file is the directory of that file;
   - the *testing directory* of a Java file which does not contain a `package` statement is the directory of that file;
   - the *testing directory* of a Java file which contains a `package pname;` statement is its parent directory.
3. remember that, as described above, the script must log using `logging.info()` all actions that modify the file system. More precisely it has to write in the log one line for each downloaded file, indicating the file name and the destination directory.

Exercise 5 - Putting everything together

Write a `main` function with no arguments, that in a loop interactively asks the user which of the four functions of the previous exercises has to be executed, and asks the arguments to be passed to the function before invoking it.
When executing the Python file as a script (as argument to `python`) the `main` function has to be invoked.

Author: Andrea Corradini & Matteo Busi

Created: 2018-12-18 Tue 17:51

Validate