AutoGrader

Joseph Kilgore

May 1, 2021

Contents

Chapter 1

Setup

1.1 Recommended Software

The autograder is a python application with git source control and LaTeX documentation. This means you will need a version of python to run this program, an IDE to work in, and some form of git for source control of the project.

1.2 Python

Python currently has numerous versions, and not all of them are compatible. Now is a good time to note that Python 2.x has been deprecated since Jan. 2020, and you should use python 3.7 for running this application. Note that you will likely need to change some environment variables in your system, and this step should be done during the python installation.

You can download python at:
https://www.python.org/downloads/

1.3 Github Desktop

Git is a version control system, and we will be discussing the version control of this particular project later, but we will need something to look at different versions, and to track changes that may be made in the future.

Github desktop gives users a simple and easy to use GUI for git (as oppose to using command line git). To download Github desktop go to:
https://desktop.github.com/

1.4 Visual Studio Code

Visual Studio Code is somewhere between a full fat IDE like Visual Studio Community, but more power than just a basic text editor. Visual Studio Code is an industry standard environment for python programming. It functions just like any other text editor/IDE. Just open the directory of your project, and you are off to the races. You can then run the python program from within VS Code

and upon first run you will be prompted to choose a python interpreter (where you can select the one you installed earlier).

To download VS Code go to:
https://code.visualstudio.com/download

1.5 LaTeX

This is completely optional and is only needed to update or modify this documentation. LaTeX is a software system for document preparation. To install it you will need a tex compiler and an editor of choice.

MikTeX compiler:
https://miktex.org/download

Texmaker editor:
https://www.xm1math.net/texmaker/

Chapter 2

Configuring Autograder

To make the autograder a more admin friendly application, a defs.py file was created to store all configurations that are available to the user for modifying how a project will be run, and where to locate the projects that will be run.

2.1 MAX_RUNTIME

MAX_RUNTIME is used for the timeout error on running any set of test cases. This is set in seconds, and if any set of parameters on a project takes longer than that period of time, the execution will be halted, and it will throw a timeout error that will be listed in the errors of the student.

2.2 NUM_PROCESSORS

This is used for multithreading the execution of projects. To turn off multithreading entirely (in the event that the autograder needs to be debugged) switch the run_multiprocessing method in grader.py with the commented code provided below it.

```
# FROM grader.py
def run_multiprocessing(studentList, inputList, gradingKey):
    global NUM_PROCESSORS
    with Pool(processes=NUM_PROCESSORS) as pool:
       pool.map(partial(gradeIndividualStudent, inputList=inputList,
         gradingKey=gradingKey) , studentList)
    #  for student in studentList:
    #      gradeIndividualStudent(student,inputList,gradingKey)
```

For additional documentation on multithreading with python see:
https://medium.com/python-experiments/parallelising-in-python-mutithreading-and-mutiprocessing-with-practical-templates-c81d593c1c49

2.3 SOLUTION_BIN_FOLDER

Put the folder path to the solution bin folder which should contain the solution .class file.

2.4 INPUT_FILE_PATH

Input to an input file path which is used for project 1. See the following chapter on Project 1 for more details.

2.5 STUDENT_FOLDER_PARENT

This is the parent folder for ALL student project folders. My recommendation for steps to extract student projects (which should have been 7zipped) is to do the following.

1. Download all projects from blackboard as a single .zip
2. Unzip the .zip to its own folder
3. In the unzipped directory should be all the student .7z. Select all of the zipped workspaces and extract all to */. This will put all the .7z into a directory with the autonamed files (this is important for students who don't name their workspace correctly, the name of these folders will be the name of the autograder output text files).

2.6 CLASS_NAME and CLASS_FILE

These two variables are used in creating the command that runs the student .class file. It is extremely important that all students name their .java to whatever this class file is or it will not be run. The autograder will search their *entire* workspace for the first .class file that matches this. This means they need to exactly match this project AND they cannot have multiple projects with the same .java file in them. They also need to make sure they have compiled their final piece of code. I'm going to assume that students will test their final code (even though we all know that probably isn't the case).

If no files are found to match in a student directory it will be listed in the console output of the grader. These folders can then be analyzed to see which students need to be modified to work.

2.7 PATH_TO_OUTPUT

This is the folder that all of the output text files will be put.

Chapter 3

Project 1

Project 1 is a connect 4 command line game. To access the code within the repo, ensure that you are on the Project 1 branch. The key note of this autograder is that it's meant for running multiple test cases from a text file and getting a particular command line output. The way this works is taking a input text file that is in the following format

```
inputs:3
0
1
2
```

Where the first line of any set of input is the number of inputs, and then followed by that many lines of input afterward. This can be repeated for as many test cases wanted. Each of these test cases will then be passed to the tested program and the output of that run will be saved.

The key work done here in the Student class is two dictionaries that map each input (literally using the input fed through to the command line) to the output and any errors that occurred. We can see that the testStudent method in the Student class does all of the heavy lifting here in terms of running projects. The key piece of code used is the subprocess package with the Popen command. This creates a process with the passed command and we open pipes for the stdin, stdout, stderr so we can interact with the student project once running.

Using these dictionaries, we compare the outputs of each student compared to the solution output and if they match then the test case was successful.

3.1 Running Projects

To run the project, all of the configuration settings in the defs.py file must be set accordingly. From there the grader.py file needs to be run. Output in the console will look like the following.

```
NO CLASS FILE AVAILABLE Smith, Joe PROJECT 1
grading student : Kilgore, Joey proj1
grading student : Hobbs, Joe proj1
```

This shows that Smith, Joe did not have a matching class file in his working directory. The other two projects were run, and should have generated text files accordingly that look like the following.

```
Kilgore, Joey proj1
AUTOGRADER SCORE : 1/1
------------------------

CORRECT OUTPUT
INPUT:
0
1
2

YOUR OUTPUT
THE SUM IS 3

SOLUTION
THE SUM IS 3

ERROR

----------------------
```

The output file has 4 sections for each test case. First, the actual list of inputs passed to the program for the test case. Second, the student output from that test case. Third, the solution output (which hopefully the student output matches this character for character). Fourth, any errors that their program output.

Chapter 4

Project 2

Project two has students read data files in and create objects with that data. Instead of passing information through stdin like Project 1, we need to copy data files into the bin directory of each student, and then we will break the output into pieces and grade students that way. To access this code make sure you are on the Project 2 branch.

In addition to setting the appropriate configuration variables in defs.py, you will also need to put the necessary files in the copyfiles directory in the autograder main directory. All files in this copyfiles directory will be copied into every single students /bin directory containing their class file.

Running the autograder is the same as for Project 1, but the way the final output is graded and compared is slightly different.

4.1 Grading Breakdown

Instead of looking at test cases (we only have 1) we will take apart the output from the program, and break it into parts that we will compare. Each line of output is broken up on '|' and then each of these parts is compared. Additionally, all whitespace is stripped out. A sample output is provided below.

```
Kilgore, Joey proj2
AUTO GRADER SCORE : 10/12
--------------------------

YOUR OUTPUT
+------------+--------------+--------------+
| SOME TABLE | WITH COLUMNS | MANY COLUMNS |
+------------+--------------+--------------+
| value1     | value2       | value 4      |
| row2       | row4         | row5         |
+------------+--------------+--------------+

SOLUTION
+------------+--------------+--------------+
| SOME TABLE | WITH COLUMNS | MANY COLUMNS |
+------------+--------------+--------------+
```

```
| value1     | value2       | value3       |
| row2       | row4         | row6         |
+------------+--------------+--------------+

ERROR

MISSED
IN YOUR LINE: | value1     | value2       | value 4      |
YOUR OUTPUT:  value 4
SHOULD BE:  value3

IN YOUR LINE: | row2       | row4         | row5         |
YOUR OUTPUT:  row5
SHOULD BE:  row6
```

There are a few things going on, but simply put there are 4 sections. A 'YOUR OUTPUT' section which displays the students output. A 'SOLUTION' which shows what the correct output should have been. An 'ERROR' section to show any errors that occurred (if there were any). Finally, a 'MISSED' section allows students to see exactly what portion of the output they messed up.

From the example we can see how the autograder breaks up the output for grading. Every '|' divides a row into chunks (each worth 1 point). In this case we can look at the point breakdown by rows in the following.

```
+------------+--------------+--------------+ (1pt)
| SOME TABLE | WITH COLUMNS | MANY COLUMNS | (3pt)
+------------+--------------+--------------+ (1pt)
| value1     | value2       | value3       | (3pt)
| row2       | row4         | row6         | (3pt)
+------------+--------------+--------------+ (1pt)
```

Every line in the output is worth atleast 1 point, and then the '|' give us more ways to break the student output to give as many points as possible. It should be noted that this will give points for a matching table headers and ending (which in this case is worth 6 of the 12 points), but this should give students a substantial number of points for getting the correct table format and a running program.

Chapter 5

similarity.py

similarity.py is a complex mathematical process for determining the similarity of two documents. This code is used to generate a csv of all the student .java files for the file name provided in defs.py. It wasn't clear during grading that this was terribly helpful. It might be more helpful to use this in conjunction with sample code related to the project that can be found on the internet.

The output is a csv file that should look something like the following.

```
SmithJoe, 0.0,0.74,0.01
HobbsJoe, 0.74,0.0,0.53
Kilgore,  0.01, 0.53,0.0
```

What we see is student's code similarity to other students on a scale of 0 (perfect match) to 1 (extremely different) and numbers as low a 0.3 are common. The columns are in the same order as the rows so we could take this into excel and copy the row labels and make them column labels in a table like the following.

	SmithJoe	HobbsJoe	Kilgore
SmithJoe	0.0	0.74	0.01
HobbsJoe	0.74	0.0	0.53
Kilgore	0.01	0.53	0.0

This is a quick way to get an immediate look students code and check for any extremely close matches. We see the diagonal should be all 0's (because it's a comparison of a student with themselves). However we can see (especially if you turn on color gradient in excel) that SmithJoe and Kilgore have extremely similar code. This is to only be used as an *indicator* for code that should be checked, and is not an end all be all metric.

For more information on the origins and math of the script see the following: https://www.geeksforgeeks.org/measuring-the-document-similarity-in-python/