Total Course Points

Stephan Kaminsky

Jan 19, 2021

CONTENTS

TotalCoursePointsFE is a Python codebase which utilizes the TotalCoursePoints API to generate a class's grades at UC Berkeley. It has a Gradescope frontend which allows students to have an updated view of how they are doing in the class.

Features:

- Local Setup to generate the roster by stitching together different rosters from UC Berkeley. It also uploads what is needed for students to Gradescope.
- A Main Setup which will grab and generate your classes data so that you have a Classroom object.
- A Main which is callable by Gradescope so that it can generate a report for the students.
- Multiple helpful grading scripts in the grading folder.
- Very customizable as you can add custom code to files/settings/ which can change how your classes assignments and other grading policies are.
- Is able to pull from a Google Sheets to pull grades and extensions.

Total Course Points

CHAPTER ONE

INSTALLATION

```
git clone https://github.com/61c-teach/TotalCoursePointsFE.git
```

Requirements: Python 3.7+.

After you have downloaded it, please modify the settings. Please see “Getting Started” for more information”.

CHAPTER

TWO

GETTING STARTED

2.1 Getting Started

2.1.1 Basic Setup

Setup with Gradescope

A quick overview of the flow of how the autograder system works. The *local_setup.py* script will generate the roster from multiple CSV files. It also will upload the required submissions so that students have a submission on Gradescope. This is run first. The next thing is *main_setup.py*. This program will use all of the prameters you have entered to generate the class data (*c.data*) file and store it at *files/data/c.data*. This script is default ran when the autograder is built so that it does not make a lot of redundant calls to the Google Sheet gradebook (if you are using that). The problem with this is you need to rebuild the autograder every time you need to update the classroom data. This is not too much of a problem as I have added a script in here *upload_and_rerun.py* which will automatically do this for you. Once the autograder has been built, the base autograder container on Gradescope will have the full classroom data. The final step is for each individual autograder to run the *main.py* script to generate a students specific information so they can see their results on the autograder. This document will go through the basics of what needs to be done to configure Total Course Points. If you would like to see more information about how to use each file, please go to the *Default Files Documentation*.

Gradescope

First thing you need to do is create an autograder assignment on Gradescope for your class. I like to name it *Total Course Points*. The next step to set up the Gradescope assignment is to add the autograder.

Once you have created it, extract the course ID and the assignment ID from the url of the assigment (*https://www.gradescope.com/courses/{COURSE_ID}/assignments/{ASSIGNMENT_ID}*). For example, if I had an assignment url of *https://www.gradescope.com/courses/123456/assignments/123456789*, the *COURSE_ID* would be *123456* and the *ASSIGNMENT_ID* would be *123456789*. Once you have those numbers, enter them into the *files/constants.py* file under *COURSE_ID* and *ASSIGNMENT_ID*.

Gradescope Deployment Autograder

The next step is to create a Gradescope Deployment Autograder zip file. This file will will be what is submitted to Gradescope to build the autograder. In the process of setting up the environment, it will also run the *main_setup.py* file which will generate your class data.

To correctly set up this file, you will first need to run to generate a deployment key for the autograder:

```
cd GDA
./gen_deploy_key.sh
```

Once you have ran that, you will see two files get created *deploy_key* (your private key) and *deploy_key.pub* (the public key for that private key). You will need to navigate to your repo on GitHub. Next, go to your settings, then click *Deploy keys*. Once there, click *Add deploy key*, enter whatever you want in the title (e.g. GDA) and then copy and paste in the contents of the *deploy_key.pub* file and click *Add key*. You do not need to click *Allow write access* and I would strongly recommend that you leave that disabled!

Next, edit the file *GDA/settings.sh* to set the *GHUSER* and *REPO* fields. *GHUSER* is the username which holds the *REPO*. For example, if I had a repo Venus in my account (ThaumicMekanism, aka the URL would be *ThaumicMekanism/Venus*), I would enter *ThaumicMekansim* for the *GHUSER* and *Venus* for the *REPO*.

Once you have done this, cd into the *GDA* directory (if you are not still in there) and run the *compress_GDA.sh* script with no parameters.

```
./compress_GDA.sh
```

This will generate a zip file in the *GDA* directory named *TotalCoursePointsFE-GDA.zip*. Please do not change the name or location as it is the script which *upload_and_rerun.py* uses (please look at the top of the file to confirm the directory and filename).

files/constants.py

This is the main file which will be referenced by a lot of other files to get the constant locations of items. The main thing you need to do is modify the *GSHEET_ASSIGNMENTS_ID* and the *GSHEET_EXTENSIONS_ID* to point to those spreadsheets if you plan on using them. If you do not plan on using them, please go to the *main_setup.py* file and remove those from it. Additionally, you will need to add the *COURES_ID* and *ASSIGNMENT_ID* of the Gradescope assignment to this.

Google Sheets Setup

You can use google sheets to enter your grades for assignments instead of leaving them as CSV files in the *files/data* directory. If you do this, you will need to go to the Google Console and create a credential which can read from the Google Sheets API. Once you have done that, download the credentials as a json file. Finally, you should put that file in the *files/input/credentials.json* file.

The next step is to create the sheet and share the sheet with the iAM account. The way TCP figures out what assignment is what is by the name of each sheet on the sheets. It does naming by the {category name}/{assignment ID}. If you have a sheet which does not have that naming scheme, it will ignore it. Also TCP requires you to have a few fields such as *Email*, *Name* (or *First Name* & *Last Name*), *SID*, *Total Points*, and *Status* (which is either *Graded* or *Missing*. It is optional to have the lateness category (*Lateness (H:M:S)*) though if you have it, it must be formatted as such: *{hours}:{minutes}:{seconds}*.

Another sheet which TCP works with is extensions. To set it up, it is the same as above though the sheet looks for different things. For this one, each sheet must be named after the category of assignments. Then you must have an *SID* column which is what TCP uses for searching for extensions. Finally, the other columns which TCP will look at are any which have matching names as the assignment IDs under that category. If it finds a match, it will read the SID and the number of extension time you gave them (default is in days though that is customizable in the assignment settings).

files/assignments.py

This is where you should add all of the assignments of your class. Please check out the Total Course Points documentation for what each assignment is.

Categories are a group of assginments which are useful for organization. You can add as many assignments to a category as you would like. You also have assignments which have many settings to customize for your class. Please check out the Total Course Points docs to view all of the options.

files/clobber.py

This is very similar to the custom grading though it is an example of how to calculate a clobber on an assignment. This function is called by *main_setup.py* You can just return early to not do any of this.

files/custom_grading.py

This file contains a function which will allow you to add custom grading to your classroom object. You do not need to do anything if you do not need to add custom grading though it can be useful if you need to make grading adjustments for students.

files/grade_bins.py

This file declares the *grade_bins* variable with the grade bins with your class. It must declare this variable as it is used in may of the other scripts. You should specify the grade letter, how many points it is worth on your grading scale, the minimum number of points for that bin, and the maximum number of points for that bin. If you set it to *None*, it will treat it as +/- infinity.

local_setup.py

This file will generate the roster and also upload the submissions to gradescope. It supports no parameters which is what you will want to do for the first time.

This file will generate your roster based off of the CalCentral roster, Gradescope roster, and the CalCentral Grade roster. It will also upload a submission for each student so they will be able to view their Total Course Points.

Note it also supports a few parameters: *regen* and *sync*. If you add regen (eg. *python3 local_setup.py regen*), this will only regenerate the roster without uploading all submissions for every student.

main_setup.py

This file is the main file which will generate the classroom data. Open it up and search for the first FIXME. You should enter your class name and ID. You can add custom messages to the classroom in here as well. This file will call all of the other scripts and dump the classroom data to *files/data/c.data*. If you would like to run it locally, you can also add the *stats* parameter when calling the *main_setup.py* script to generate a pretty graph of your classroom statistics based off of the grade bins you have.

upload_and_rerun.py

This is the final step which will rerun your autograder and regrade all student submissions. You do not need to modify anything in this file unless you customized the names of the GDA zip file. All you need to do is run this file and it will upload the zip file, rebuild the autograder, and finally rerun all submissions so that your students can view their grades.

Setup without Gradescope

The current infrastructure requires a Gradescope roster. This is only required by the *local_setup.py* file. In the future, it will check to see if the file does not exist and skip those steps if that is true. Other than this, the steps after *local_setup.py* are the same after the Gradescope setup.

CHAPTER

THREE

DEFAULT FILES DOCUMENTATION

3.1 Default Files Documentation

3.1.1 *local_setup.py*

This file will generate the roster. It will build the roster based off of the *files/input/calcentral_roster.csv*, *files/input/calcentral_grade_roster.csv*, and *gs_roster.csv* to generate the roster. It will place the roster at *files/roster.csv*. It will then upload the 'submissions' to the Total Course Points Gradescope assignment for each student so they can view how they are doing in your course.

This file has some parameters which it takes in: *regen* and *sync*:

regen

This will only regenerate the *files/roster.csv* file. It will not attempt to upload anything to Gradescope.

sync

This option will only upload submissions to students who are in the roster but do not have any submission. This is very useful for if you need to give some students submissions but you do not want other students to have a record of previous runs of TCP.

3.1.2 *main_setup.py*

This file will generate the classroom data based off how you set up the class. It is default set to pull grades from a Google Sheet though it first checks to see if you have a grade csv in the *files/inputs* directory first.

stats

If you add this parameter, it will also print out the bins and other class statistics when it builds the data. It used to always do this though Gradescope was having issues where builds would hang so I changed it to a parameter.

3.1.3 *main.py*

This file will generate the results.json file which Gradescope understands. You can also enter a student's SID as a parameter to generate a specific students output. This is useful as you can use the GradescopeBase to run the json file to view the submission. If you do not enter a parameter, it will default to searching for the input.json file.

3.1.4 *upload_and_rerun.py*

This file will upload the GDA to Gradescope to rebuild the autograder and then rerun all students submissions so they can see how they are doing in the class.

3.1.5 *files/*

roster.csv

This is the roster of your class. The roster has multiple columns which Total Course Points expects.

Name

Type: String

This is the name of the student.

SID

Type: String

This is the student's ID. It must be unique as this is the primary grouping key.

Email

Type: String

This is the student's email. This should also be unique though is the second grouping key though is not used by everything.

InCanvas

Type: Boolean

This parameter tells you if the student is taking the class. This is required as you may have non-students taking the class but not want to include them into your stats.

ForGrade

Type: String

This is the grading option which the student is taking the class for. TCP only supports a few grading options such as GRD, CPN, DPN, EPN, ESU.

Incomplete

Type: Boolean

This is another method of designating that a student is taking an incomplete in your class. I recommend that you do not use this and instead use the post processing to set incompletes.

constants.py

This file contains a lot of constants which many other files use.

settings

This directory contains the main settings of your class.

CHAPTER

FOUR

HOW TO CONTRIBUTE

Please make sure to take a moment and read the Code of Conduct.

4.1 Report Issues

Please report bugs and suggest features via the GitHub Issues.

Before opening an issue, search the tracker for possible duplicates. If you find a duplicate, please add a comment saying that you encountered the problem as well.

4.2 Contribute code

Please make sure to read the Contributing Guide before making a pull request.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search