SETUP

There is a short one time process in order to get everything working. The first step is you need to make sure that you have the canvas api installed to your computer so that python can use it. Most editors nowadays already have pip installed so this terminal command should do it

pip install canvas api

If that returns an error, you may need to install pip first. It can sometimes be a complicated process so we have linked a guide to help.

https://pip.pypa.io/en/stable/installation/

The way the grader is built makes it so that very little code must be edited. All customization will be done through the JSON config files. In order to access your canvas course however, the “main.py” file requires your courses’ “Canvas API URL” and its “API KEY”. With these it will be able to pull your assignments and grade them automatically.

Canvas API URL:

- This can be found near the top of the main file. You will need to use the instructure link specific to your district (For example, alpine districts link is “alpine.instructure.com”)

Canvas API KEY

- Your API key is needed to authorize actions done to your canvas course. As this grader is a local file without any access to the internet other than canvas, it is completely safe to put your key in.
- You would need to fill out a request form in order to get an API key as it is very valuable information
- The link below explains the process to request an API key. https://kb.iu.edu/d/aaja

Json Documentation

In order for the code to understand what the specifications are for each assignment, you need to make a config JSON file that specifies how you want this to be graded, how many points each function awards on success, and what inputs and outputs to expect. We provided a default JSON to use as a blueprint when making your own. You can easily just copy it and rename it. Then edit it for your specific use.

It is also wise to only keep config files for assignments you want graded in the "configs" folder. The grader will grade **every** assignment in configs. So only keep the configs that you want graded.

The JSON is split into three categories: Canvas, Settings, and Tests.

Canvas

COURSE_ID:

- This course id will let the grader know what course to grade. You can find this by looking at the url of the course's home page. It should be a series of numbers at the very end of the url.

ASSIGNMENT_ID:

- This id is specific to each assignment, you would presumably be making a new config file for each canvas assignment you want to be autograded.
- The assignment specific id can be found by looking at the url of the assignment. It should be at the very end after "module_item_id=". Simply use that number as the assignment ID.

Settings

In order for the grading process to be more customizable, we have provided a multitude of settings for grading. Below are the settings and how they will affect the grading process.

Most settings will be dictated by a “true” or “false” statement following the setting in the json.

IMPORTANT TO KEEP IN MIND

- Canvas is very flawed in how it handles assignments. Once something is submitted and graded, canvas will view it as an unsubmitted new attempt on future iterations.
- We have implemented a warning, in which the grader won’t regrade assignments that are not NEW assignments.
- This shouldn’t interfere too much with the grading process, but it may be helpful to be aware of this critical flaw in the canvas api.

IGNORE_UNSUBMITTED

- If this setting is set to “true”, the grader will not grade a student's assignment if it’s missing. It will ignore it, and it will need to be graded later once the student has submitted it.
- If this setting is set to “false”, the grader will give any missing assignment a “0”, along with any already submitted assignments regardless of the grade given. Due
- It’s highly recommended to just keep it at true.

IGNORE_ALREADY_GRADED

- If this setting is set to “true”, the grader will skip assignments that have already been given a grade.
- If this setting is set to “false”, the grader will grade assignments that have already been given a grade.

ONLY_GRADE_STUDENTS

- If you put any student names in brackets, the grader will ignore all students but them.
- This can make you grade only a select few students.
- In order to put students in the list, you must write the student’s name
- Ex: “ONLY_GRADE_STUDENTS”: [“John Addams”, “Paul Mcartney”]
- This ignores and over rules all other settings.

DONT_GRADE_STUDENTS

- When a student is in the brackets, the grader will skip over them.
- You can add students into the brackets the same as you would for ONLY_GRADE_STUDENTS

REGRADE_STUDENTS

- Students placed in the brackets will be regraded, along with any assignment that has not been graded yet.

Making Tests

The tests you make in the config file are what ultimately gives your students their grade. If you are wanting to make a new test case, it is recommended that you use our default config json as a blueprint to help you write yours. The syntax is really specific for the grader to read. If you are confused, simply reference the example json.

There are multiple variables that need to be specified for each list, they are as follows

POINTS

- This will tell the grader how many points to award when a function passes its required test.
- The total points of all the functions should be the total points for the assignment given, that way if a student passes all tests, their assignment will be given 100%.

FUNCTION_NAME

- This is the name of the function that you are wanting to grade.
- It is very important that all students use the same names for their functions that are being graded. If a student has a different name, the grader won't be able to test it, giving that student a 0 for that portion
- The names of the function need to be in quotations

INPUTS

- These are the inputs that you are wanting to test for each function.
- Each test input must be within brackets, separated by commas (assuming a function requires more than one input).
- Integer and float/double values don't need quotation marks, while strings do. Other data types will need their specified syntax.

EXPECTED_OUTPUTS

- These are the outputs that you are wanting the inputs to return.
- The outputs will be separated by commas.
- For example, if in inputs I wrote [[3, 2], [3, 5]] for the function "subtract". The I would intend would be 1 (3 - 2) and -2 (3 - 5). To write these expected outputs, they would need to be formatted as [1, -2].
- If the expected outputs match the outputs that the student's function returns, they will be rewarded with the points specified earlier in the test

OUTPUT_TYPE

- This specifies if the function will return the outputs by using the "return" or by printing the output.
- If you are wanting the output type to be returned, write "return" in quotations.
- If you want to test for printed output, write "print" in quotations.

API Documentation

Through the way we built this, there is very little need to understand or edit the code. It's a pretty tight system, and all customization is done in the json config files. However, api documentation could be beneficial if someone were wanting a more personalized experience.

These are the functions that are used throughout the project.

get_config_files

- This function just grabs all of the files in the "configs" folder

get_submissions

- This function will first go to the specific assignment and course specified and grab all submissions.
- Second, the function tests all of the submissions, unless a rule set in the config states otherwise.
- Third, it will grab the student's submission and use the "check_submission" function to grade it. If it is not submitted (unless the rule to skip unsubmitted assignments is true) it will give that assignment a 0.

check_submission

- This function will go through every test case specified in the config json, then it will call "check_return" to see what score the test got.
- It will add each score together, then push the grade to the submission through "push_grade"

get_student_name

- This function simply gets the name of a student through their id and course.

run_function

- This function will run the student's submitted function and store the output to test later.
- If the arguments the function receives are invalid, an error will be thrown and nothing will be returned.

check_return

- This function will check if a test passes or not.

push_grade

- This will change the student's submission score and add a comment telling them what their grade and score is. It will then print a message indicating success.

get_paginated_list_length

- Returns the length of a paginated list. There is no length attribute for them so we had to write this just to help

progress_bar

- This will print a progress bar indicating the percentage of submissions completed

main

- The main function will first, call "get_config_files" to gather all of the config files
- Second, it will go through every config file and grade all assignments specified by them.