A thick, dark olive-green vertical bar runs along the left edge of the page. To its right, a red arrow-shaped banner points to the right, containing the date. Below the banner, several thin, curved lines in shades of olive and grey sweep upwards from the bottom left corner.

11/28/2016

Autograder for MATLAB

Documentation

CS 1371 Software Development Team
GEORGIA INSTITUTE OF TECHNOLOGY

Contents

Introduction	3
Section I: The Basics.....	4
Overview	4
Installation:	4
Running the Autograder	8
Section II: Advanced Settings.....	11
Overview	11
Section III: Troubleshooting.....	12
Overview:	12
Environment:.....	12
Starting up:	12
Running:.....	12
Shutting Down:.....	12
User:.....	13
Starting up:	13
Running:.....	13
Section IV: Function Documentation	14
Overview:	14
Functional Diagram:.....	15
Function: base64img.m.....	17
Function: compare.m	18
Function: comparePlots.m	19
Function: convertSupportingFiles.m	20
Function: getDirectoryContents.m	21
Function: getFeedback.m	22
Function: getGradebook.m	23
Function: getInputVariables.m.....	24
Function: getMessages.m.....	25
Function: getProblemFeedback.m	26
Function: getRubric.m	27
Function: getSettings.m	28
Function: getStudentIds.m.....	29

Function: gradeStudentSubmissions.m	30
Function: gradeSubmission.m	31
Function: handleStudentTimeouts.m	32
Function: img2mat.m	33
Function: isFolderEmpty.m	34
Function: isGradesCsvFormatDifferent.m	35
Function: isVerbose.m	36
Function: lint.m	37
Function: loadjson.m	38
Function: loadRubric.m	39
Function: overrideBannedFunctions.m	40
Function: parseTestCase.m	41
Function: readGradesCsv.m	42
Function: runAutograder.m	43
Function: runSolutions.m	44
Function: runSubmission.m	45
Function: runTestCase.m	46
Function: unzipFile.m	47
Function: uploadFile.m	48
Function: uploadFilesToServer.m	49
Function: uploadHomeworkGeneratorFilesToServer.m	50
Function: uploadStudentFilesToServer.m	51
Function: visualizeValue.m	52
Function: writeGradesCsv.m	53
Function: xls2mat.m	54
Object: Rubric	55
Object: Gradebook	56
Object: Student	57

Introduction

This is the official documentation for the autograder used in the MATLAB CS 1371 class at Georgia Tech. The autograder is, essentially, a way for MATLAB TAs to focus less on *grading* the homework and more on *creating* it. While its internals can be quite complex, using it is surprisingly straightforward.

This documentation is broken up into several sections. It can be helpful to use the below section guides to help you decide where you need to look:

Section One: The Basics deals with the initial steps; installing the autograder, ensuring that your files are in the correct places, what to download (and where to download it to), and culminates in a mock run of the autograder. If you're new to the autograder, you should start here.

Section Two: Advanced Settings shows special options you can use to streamline the process; this section also has some general observations about the autograder. If you'd like to see how it works, look at section four. If you're new to the autograder, this can be useful as well.

Section Three: Troubleshooting is designed to help you when things go wrong. While there can be a near-infinite number of reasons for the autograder failing, here you'll find listed some of the most common reasons for failure, and how to fix them. If you're experienced with the autograder, this is likely the place you'll need.

Section Four: How it Works explains how all of the included files work. This section can be especially useful if you are trying to add a feature or fix a bug. In essence, it's a reference section for the autograder. If you're a developer *working* on the autograder, this section can be invaluable.

Thanks for using the autograder, and Happy Coding!

The Software Development Team

Section I: The Basics

Overview

Using the autograder is designed to be painless; in essence, once you start the autograder, you should be able to simply start it up and let it run. Before you can do that, however, you'll need to install it! Thankfully, installation is mostly a breeze.

Installation:

Installation consists of three steps: Downloading MATLAB files, checking your Operating System, and downloading the necessary auxiliary files.

Downloading MATLAB files:

To do this, you'll need to have access to the autograder repository on GitHub. Now, there's much more to git than we could ever cover in this manual alone, but here we will show a few of the operations needed for this installation. Getting access to the repository simply requires asking the Software Development STA for access. Keep in mind that your access is through your Georgia Tech Credentials!

Git Commands:

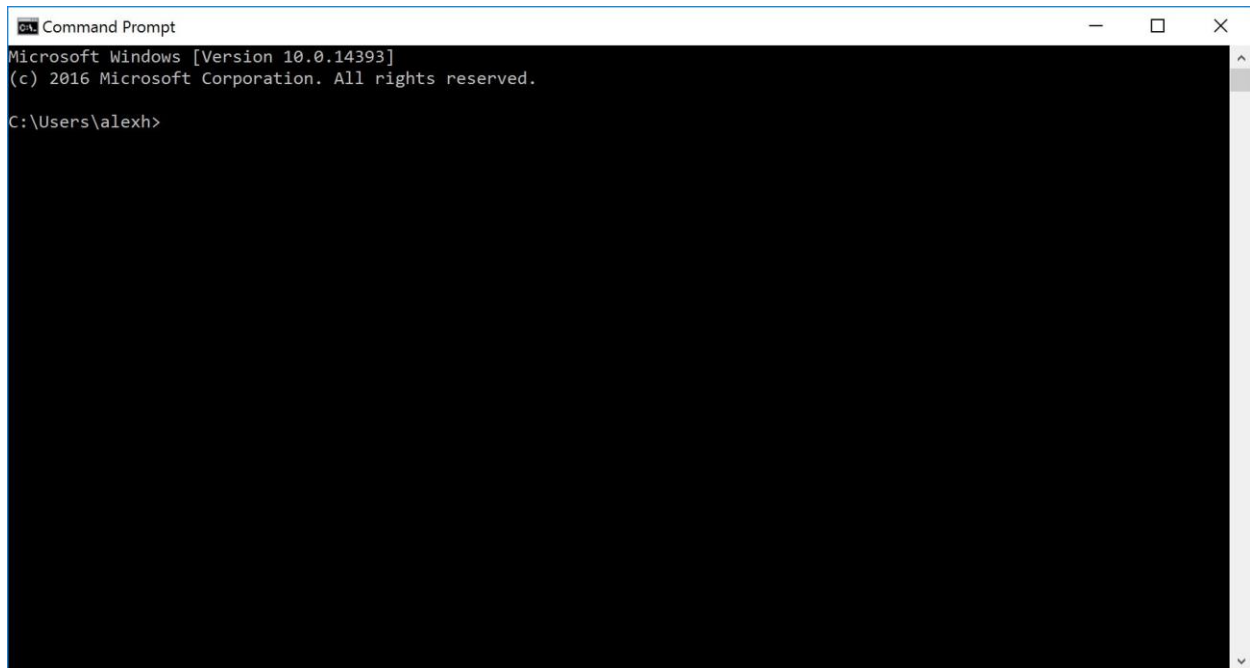
Clone: Use this to *clone* the repository somewhere on your personal computer. Wherever you choose to clone this into is fine, but keep in mind that this is where it will *always* have to live; to change the location, you'll have to clone all over again.

Pull: Use this to get the most recent set of files from the repository. In essence, this is how you check for updates.

So, now that we know about cloning and pulling, let's install the autograder.

First step is to decide where you want to store these files. You're going to want this to be a permanent place; usually the *Documents* folder is a good place to start. For this example, I'll be choosing the Documents Folder of my PC.

Then, you'll need to open your command prompt or terminal. For the purposes of this example, I'll be using Command Prompt, but you can elect to use whatever you have.

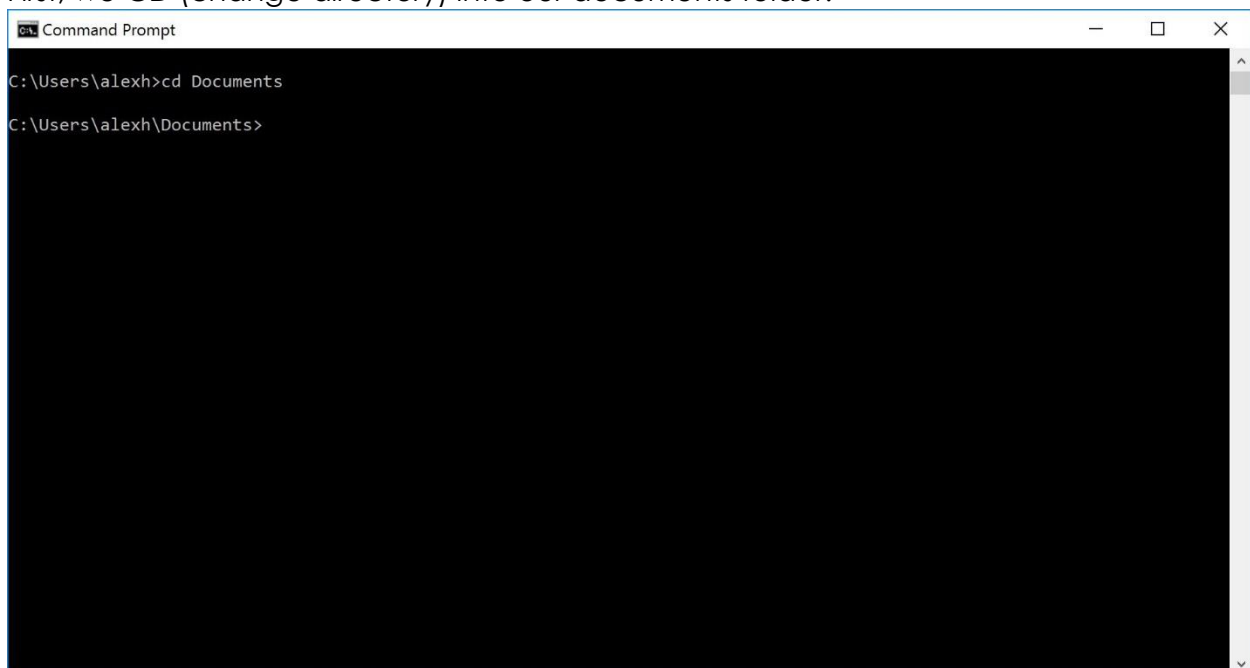


```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\alexh>
```

1 The Command Prompt

First, we CD (change directory) into our documents folder:

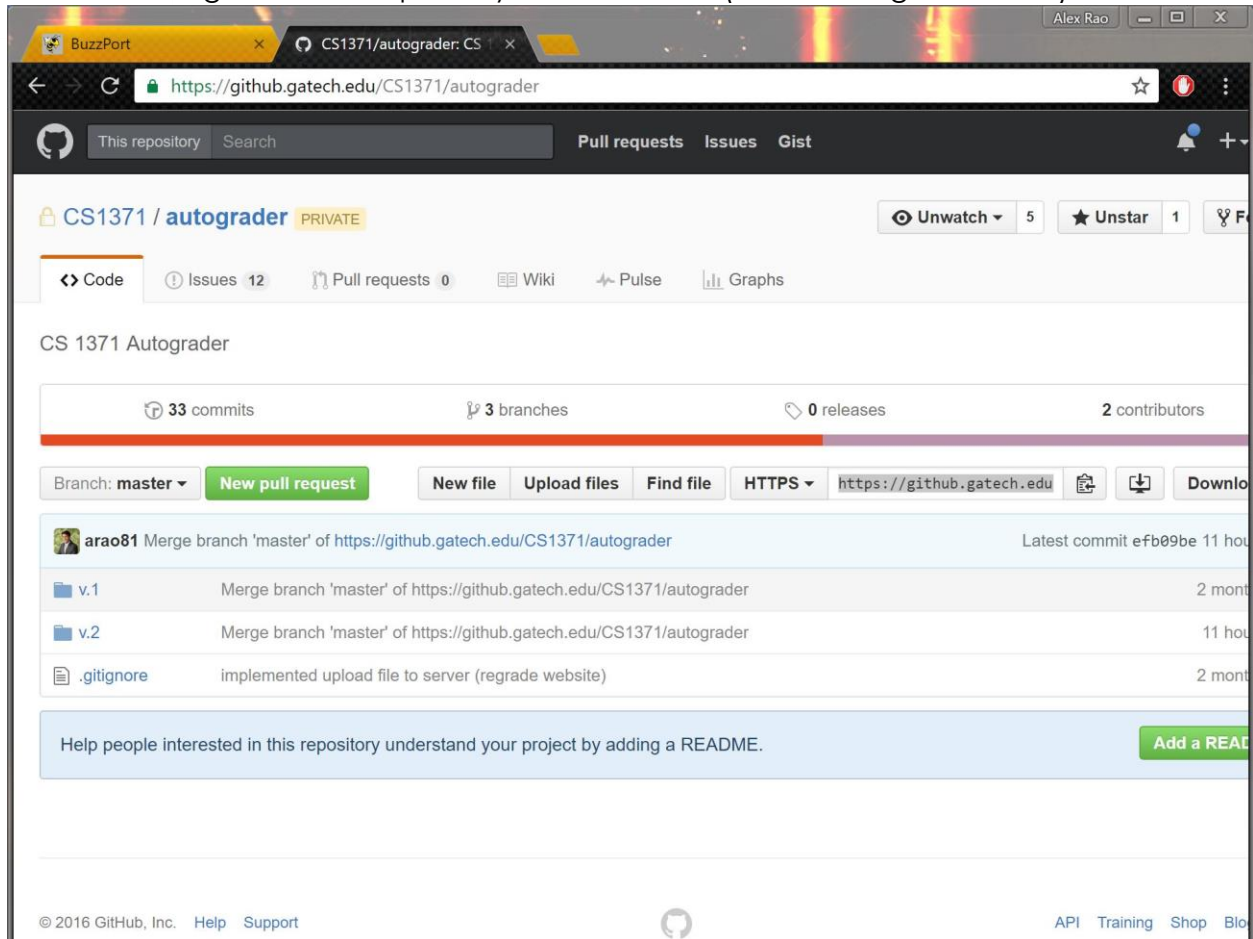


```
Command Prompt

C:\Users\alexh>cd Documents
C:\Users\alexh\Documents>
```

2 Change Directory into the Documents folder. Command Prompt is Case Insensitive!

Then, we navigate to the repository in our browser (I used Google Chrome):

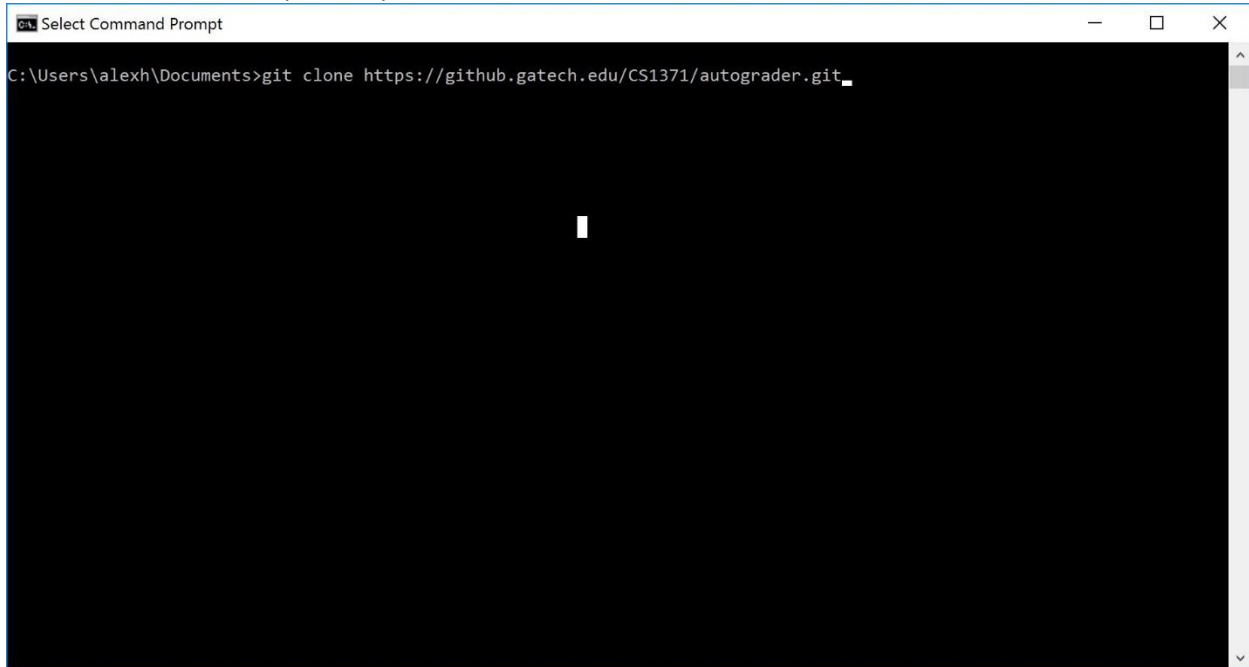


3 The repository on GitHub. Notice the button in the top right hand corner that looks like a clipboard!

Once we are here, we copy the HTTPS link for our Repository (If you click the clipboard icon, it will automatically be copied to your clipboard!). Then in command prompt, we type

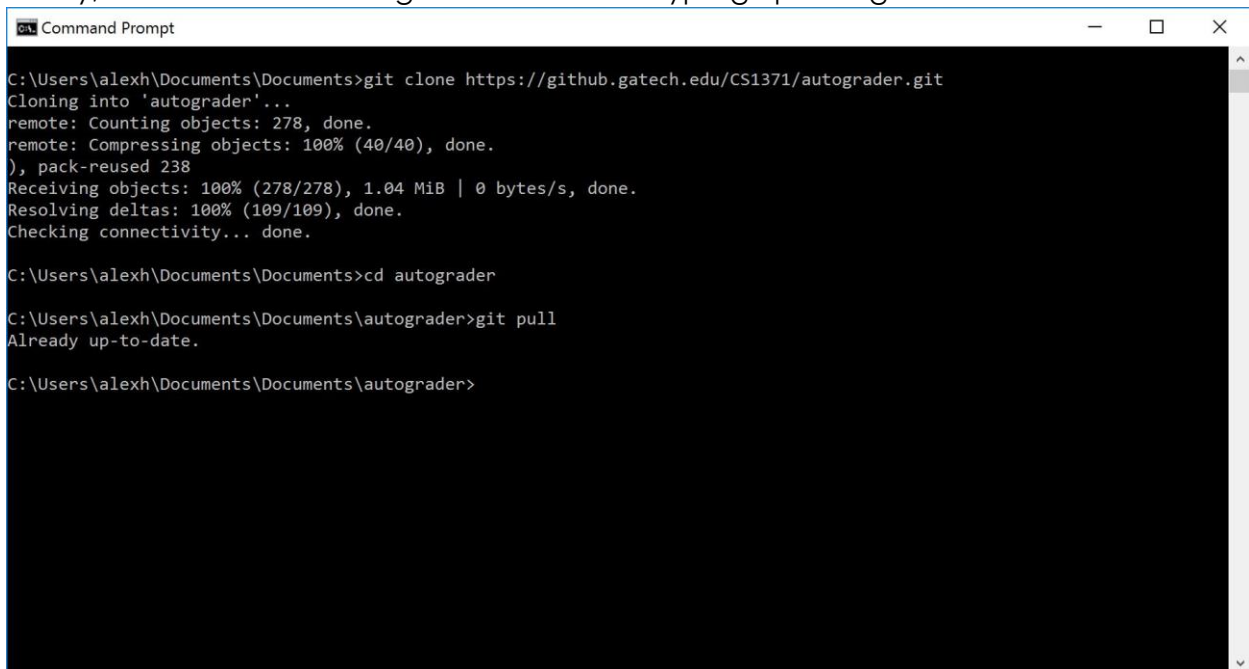
git clone <https://github.gatech.edu/CS1371/autograder.git>

This will clone the repository into our Documents folder:



4 Cloning the GIT repository. Note that it ends in .git

Finally, we CD into the *autograder* folder and type git pull to get the latest files:



5 Note that GIT tells you it's all up to date

Now that we've downloaded MATLAB files, we need to determine what operating system you're on.

If you're on Macintosh or Linux, you're done! Assuming you've set up MATLAB correctly, you should be able to continue on with no problems.

If you're on Windows, you'll need to install 7-Zip Unzip utility.

Running the Autograder

Hopefully, what you've just done is the hardest part of the journey. Now, you can sit back, relax, and let MATLAB do the heavy lifting. Before we run the autograder, however, we need to get the files. Here's how to do that:

Student Files

This guide is for use with T-Square; to download student files, go to the CS 1371 page (NOT the TA page!) and navigate to *Assignments*.

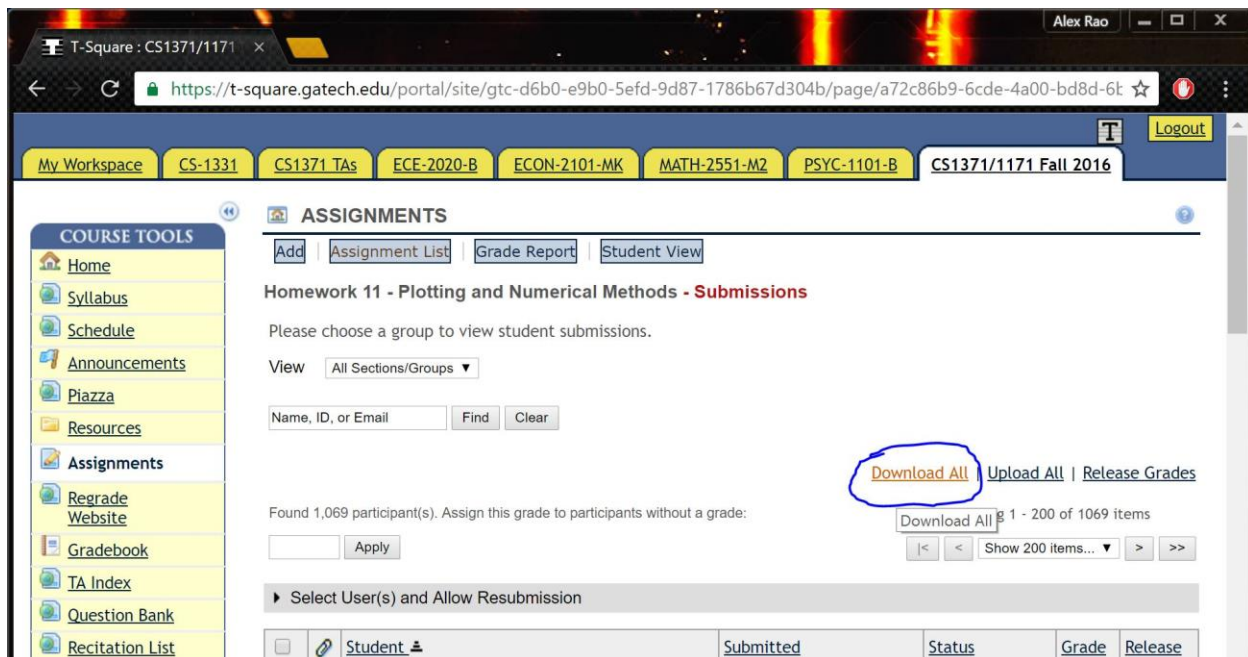
Then, click the *number* under In/New; **DO NOT CLICK ON THE ACTUAL ASSIGNMENT!**

The screenshot shows the T-Square portal for CS1371/1171. The 'ASSIGNMENTS' section is active, displaying a table of assignments. The first assignment, 'Homework 11 - Plotting and Numerical Methods', is highlighted in yellow. In the 'In / New' column, the value '38/38' is circled in blue. The 'Scale' column shows '0-100.0'.

Assignment title	For	Status	Open	Due	In / New	Scale
Homework 11 - Plotting and Numerical Methods	site	Open	Oct 29, 2016 12:00 am	Nov 4, 2016 8:00 pm	38/38	0-100.0

6 Notice that you click the *NUMBER* link; **NOT THE ASSIGNMENT!**

Finally, click Download All and then in the resulting screen check "All:"

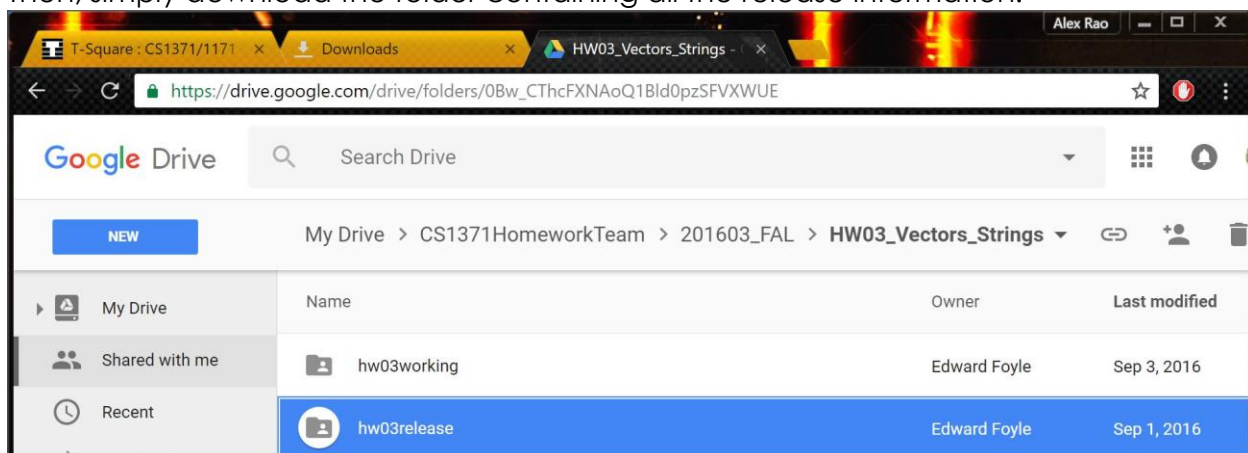


7 The Download All Link

Place the bulk_download.zip file in a central, easy-to-access place (usually the Desktop is sufficient).

Next, you'll need to download the HW## folder. To do this, you'll need access to the Homework Team Drive; The software development STA should have this readily available.

Then, simply download the folder containing all the release information:



8 hw##release is the folder you want to download; note the folder path!

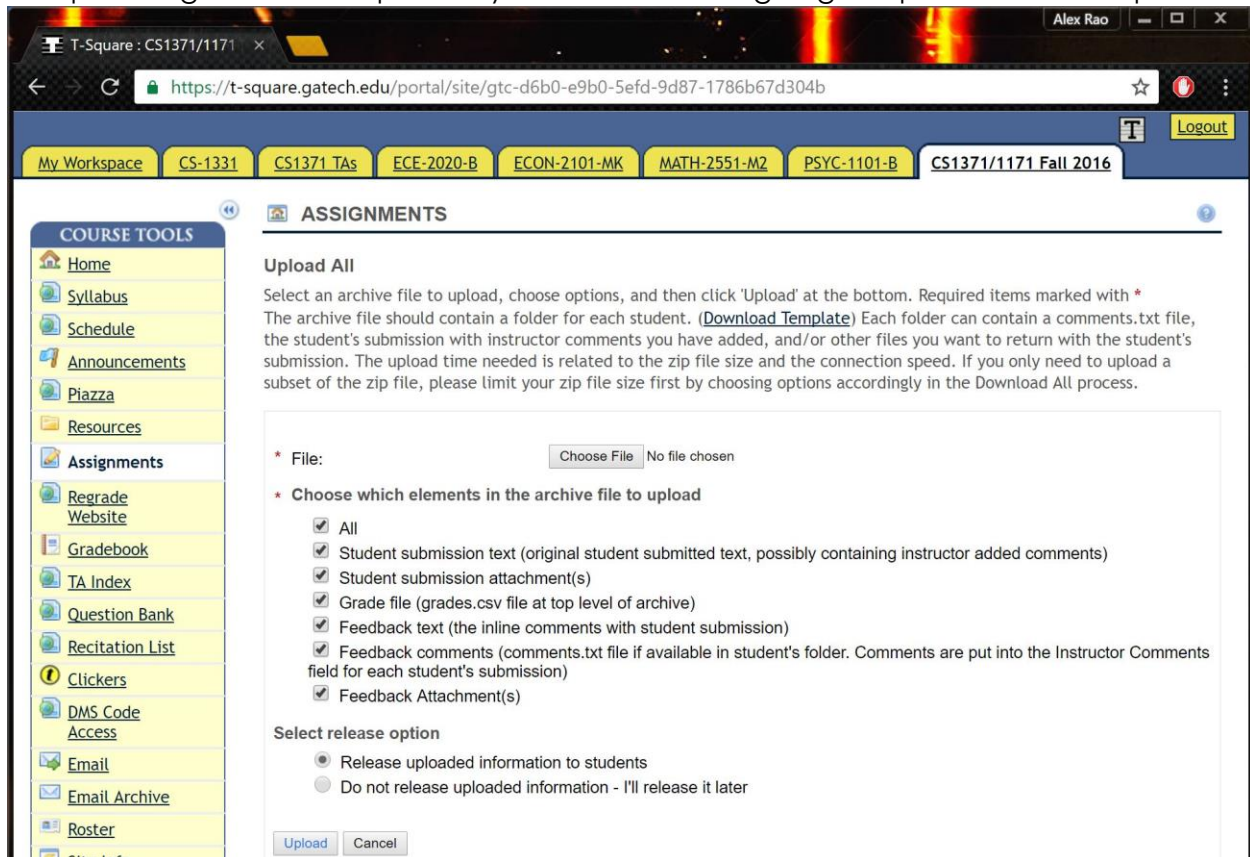
Put this folder in the same place as the bulk download.

Finally, make a new, *Empty* folder somewhere; does not matter where (usually, as before, the desktop is sufficient). Now you're ready to run the autograder!

Here's how:

First, navigate to the autograder\v.2 folder in MATLAB. This manual assumes you have at least basic knowledge of MATLAB, so I am not going to explain how to do this.

Then, simply type runAutograder – then you'll need to select (in this order): the bulk_download.zip file, the hw##release.zip file, and your newly created empty folder, and then sit back and let the autograder run! It'll publish the newly-created .zip folder for uploading back to T-Square in your folder. You're going to upload this to T-Square:



9 After you click Upload All, You'll see this screen

And that's it! Make sure you release grades to students!

Section II: Advanced Settings

Overview

This section is all about customizing the autograder, and hopefully soon, this section will be much broader. As of now, there is only one option you can use to speed up your time; if you give (in this order) the bulk download path, the homework zip file path, and the new folder path as string arguments, you won't be asked to locate them later.

Section III: Troubleshooting

Overview:

This section primarily explores where the autograder can go wrong. Issues are listed in sections: Autograder, User, and Other. Environment issues are issues with the applications you have (or don't have) installed on your system. User issues are issues with how you are trying to use the autograder. Other issues are issues that don't cleanly fit in with these categories!

Environment:

Starting up:

- 1) I receive a warning saying my folder isn't empty, even though I know for a fact that it is!
 - a. Most operating systems have a hidden file to delegate Folder Attributes; often, this file is *in* the folder itself. Macintosh's .DS_Store file comes to mind here. Regardless, this is not something you should be too worried about!
- 2) I can't start the autograder! Help!
 - a. Make sure of these things first:
 - i. You have the correct GIT version of the autograder (use git pull if you're unsure about this)
 - ii. You are in the *autograder* folder (autograder\v.2) in MATLAB
 - iii. You have the parallel toolbox (this should have been installed by default)
 - b. If you still can't run the autograder, try cloning the autograder again.

Running:

- 1) The number of names in grades.csv doesn't match the number of extracted folders in bulk download!
 - a. This is a peculiar issue, and you'll notice that the name of any persons in the grades.csv file but NOT in the set of folders has a forbidden character (usually the "?" character). To get around this, make sure the student's name is convertible to ASCII!
 - b. As a worst-case scenario, you could simply delete the row of the offending student; do this as your last solution, however, because this inconsistency can cause problems.
 - c. Make sure you do NOT edit the CSV file with Excel! When you save it, Excel will slightly change the format. Use your favorite text editor (such as Sublime) instead.

Shutting Down:

- 1) What is this parallel pool stuff and why is it stopping me from shutting down MATLAB?
 - a. Parallel pool allows a MATLAB thread to monitor another MATLAB thread; in essence, it can make multiple jobs happen at the same time. Without getting too technical, this makes MATLAB need more structured resources

that can take a while to shut down. Before shutting down MATLAB, be sure to shut down the parallel computing pool.

User:

Starting up:

- 1) I can't find the zip folder!
 - a. Make sure you downloaded the zip folders from T Square and Google Drive. Do NOT simply download the assignment!
- 2) runAutograder doesn't work!!
 - a. Look at the first question under *Environment*.

Running:

- 1) It doesn't accept the Homework folder as my bulk download!
 - a. Remember: *The Autograder unzips everything for you*. You should NEVER unzip the folder prior to running the Autograder. Furthermore, you shouldn't be downloading anything that isn't a ZIP file!
 - b. Make SURE you download the correct files! If you're unsure, refer to Section I: Running the Autograder for more information.
- 2) The Autograder keeps erroring on me!
 - a. First, make sure your ZIP files are correct; if they're wrong, nothing will work.
 - b. If all is well, then try reading the error message; these messages are usually very descriptive.
 - c. Also, see *Environment*

Section IV: Function & Object Documentation

Overview:

This section is geared towards members of the Software Development team *specifically working on the Autograder*; it isn't meant to be useful to anyone else.

The way functions are structured is as follows:

Function: <Name>:

Inputs: <Inputs>

Outputs: <Outputs>

Specific Uses (if any): <Uses>

Description: <Description>

An * means that the input is not necessarily required.

Note that similar information can be found by using the *help* functionality of MATLAB.

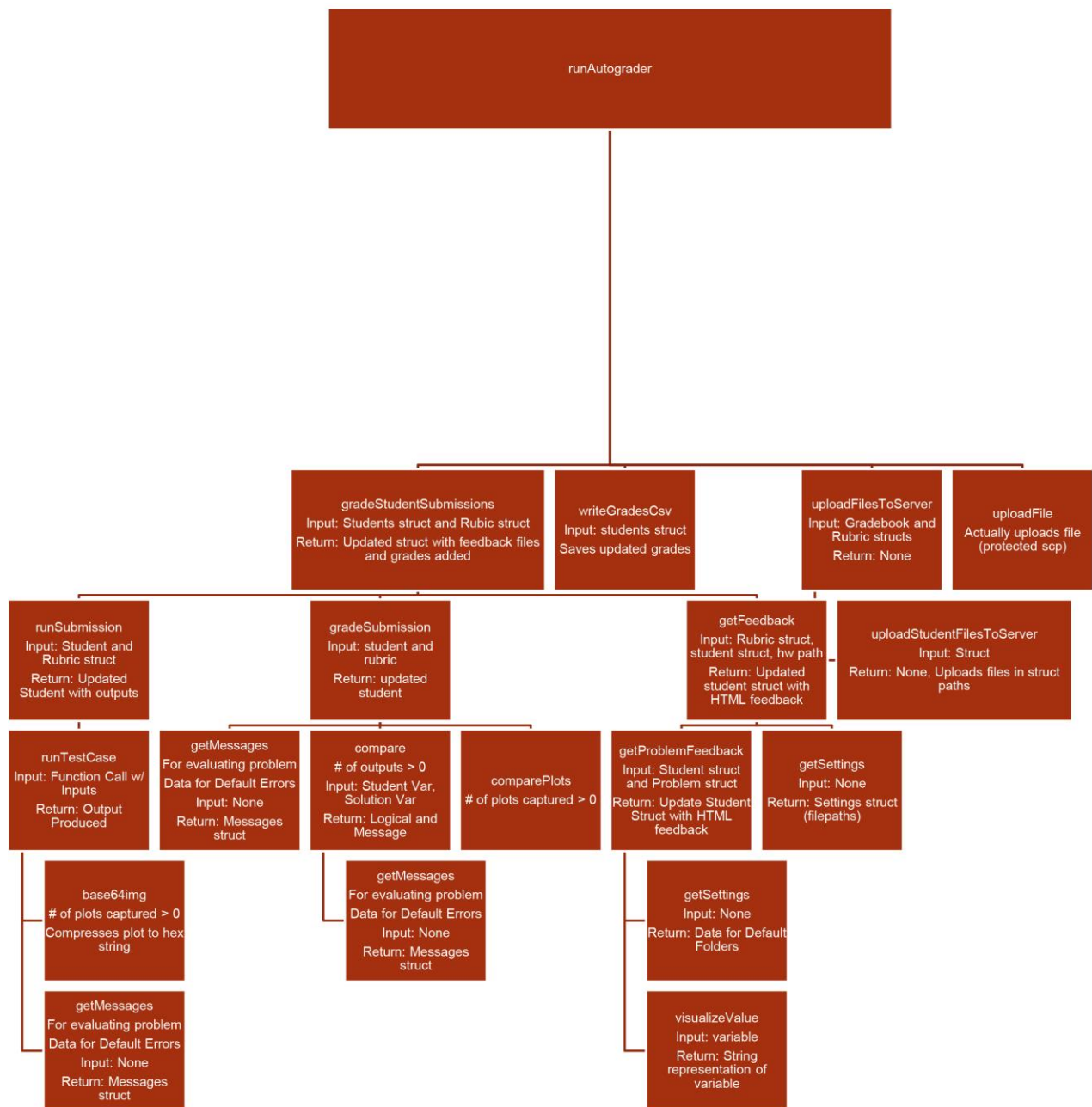
Objects are structured in a similar way:

Object: <Name>:

Description: <Description>

Functional Diagram:





Function: `base64img.m`

Inputs:

- `fig*` - a figure handle representing the figure to save as an image.
- `dpi*` - a double representing the dots per inch (resolution) to use.

Outputs: `base64string` - a string representing the image.

Specific Uses: `runTestCase.m`

Description: Given a figure handle, `base64img` will use the Apache Picture Codec to compress the image. This is a lossy conversion, so down sampling does happen. The resulting string is somewhat like a hash; while larger images will necessitate larger strings, a small difference in the string can mean a large difference in the picture, and a small difference in the picture can mean a large difference in strings. If no arguments are given, the `gcf` (`getCurrentFigure`) command is used to get the figure; if the `dpi` is not given, a default value of 75 is used.

Function: `compare.m`

Inputs:

- `studentAnswer` - a variable data type that represents the student's answer.
- `solutionAnswer` - a variable data that represents the solution answer.
- `isFile` - a logical that is true when files are being compared.

Outputs:

- `isEqual` - a logical of whether the answers are equal or not.
- `message` - a string representing a message about the comparison.

Specific Uses:

- `comparePlots.m`
- `gradeSubmission.m`

Description: `compare` will take in arguments of variable data types; one can think of `compare` as an overloaded form of `isequal()` - it will give a true or false; if it's true, the answers are the same; if it's false, the answers are different. This does not say by *how much* the arguments differ; only that they differ.

Function: `comparePlots.m`

Inputs:

- `studentPlot` - a structure of the student plot.
- `solutionPlot` - a structure of the solution plot.

Outputs: `plotComparisonResult` - a structure representing the results.

Specific Uses: `gradeSubmission.m`

Description: `comparePlots` compares two plots. These plots are given as structures, with a variable amount of fields representing `XData`, `YData`, etc. The result has the same fields as the two input plots; if these fields differ, `comparePlots` might throw an error. This is considered a bug, not a feature, and will be fixed in a future version.

Function: `convertSupportingFiles.m`

Inputs: `folderPath` - a path to the supporting files folder.

Outputs: None

Specific Uses: `getRubric.m`

Description: Given a supporting file (of either excel format or image format), `convertSupportingFiles` will generate the corresponding .mat file (so that, when test cases are run, they can always load a .mat regardless of original data storage type).

Function: `getDirectoryContents.m`

Inputs:

- `folderPath` - a path to the specified directory.
- `keepFolders` - a logical; if true, folders will be included in the output.
- `keepFiles` - a logical; if true, files will be included in the output.

Outputs: `directoryContents` - a structure containing the information on files and folders in the given directory.

Specific Uses:

- `convertSupportingFiles.m`
- `isFolderEmpty.m`
- `lint.m`
- `runSubmission.m`
- `runTestCase.m`
- `unzipFile.m`
- `uploadHomeworkGeneratorFilesToServer.m`
- `uploadStudentFilesToServer.m`

Description: Given a folder path, `getDirectoryContents` will return a structure array that has information for all of the files and folders (if you so choose) in that directory. Specifically, the structure has fields `name`, `folder`, `date`, `bytes`, `isdir`, and `datenum`; Hidden Folders are excluded on all operating systems (that is, the "." and "\$" are both removed).

Function: `getFeedback.m`

Inputs:

- `rubric` – the rubric object.
- `student` – the student object.
- `homeworkFolderPath` - a string representing the path to the student's submission folder and `grades.csv`.

Outputs: `student` - a structure representing that student.

Specific Uses: `gradeStudentSubmissions.m`

Description: Given the rubric, the structure of the student's answers, and the folder path for their files, `getFeedback` will return the html representation of their feedback, creating the skeleton of the HTML to be given as `feedback.html`.

Function: `getGradebook.m`

Inputs:

- `homeworkZipFilePath` - A string representing the path to homework .zip file. This is the file downloaded from T-Square.
- `destinationFolderPath` - A string representing the path to the working directory.

Outputs: `gradebook` – the gradebook object.

Specific Uses: `runAutograder.m`

Description: Given the zip file path and your working directory folder path, `getGradebook` will unzip the file and returns the gradebook (see this object's documentation).

Function: `getInputVariables.m`

Inputs: `functionCall` - a string representing the complete call for a test case.

Outputs: `inputVariables` - a cell array of strings, representing the input variables.

Specific Uses: `parseTestCase.m`

Description: `getInputVariables` gets the input variables from the function call. It is relatively straightforward in how it does this, using `strsplit`. The cell array output is in the same order as the function call; first input is the first entry in the cell array.

Function: `getMessages.m`

Inputs: NONE

Outputs: `messages` - a structure with all the default error messages.

Specific Uses:

- `compare.m`
- `gradeSubmission.m`
- `runTestCase.m`

Description: This returns the default error message to use if a student causes the specified error. It is a convenience method; instead of changing the class mismatch error everywhere you use it, just change it here.

Function: `getProblemFeedback.m`

Inputs:

- `problem` - A structure representing a specific homework problem; see the documentation for this object.
- `student` - A structure representing the student; see the documentation for this object.
- `problemNumber` - a double representing what problem we are on.

Outputs: `student` - A structure representing the student, updated with feedback for the specified problem.

Specific Uses: `getFeedback.m`

Description: Given a problem and a student's results, as well as the problem number, `getProblemFeedback` will return the student structure updated with an HTML representation of the breakdown for that specific problem. This HTML is used by `getFeedback` to create the `feedback.html` for that student. It uses MATLAB's `visDiff` for text and Excel file comparison.

Function: `getRubric.m`

Inputs:

- `rubricZipFilePath` (char) - path to the rubric .zip file.
- `destinationFolderPath` (char) - path to the working directory.
- `isResubmission` (logical) - whether the homework being graded is the original or the resubmission.

Outputs: `rubric` – the rubric object.

Specific Uses: `runAutograder.m`

Description: Prepares the rubric structure that contains all the information about the homework that week in terms of solution outputs and supporting folders

Function: `getSettings.m`

Inputs: None

Outputs: `settings` - a struct containing the settings for the autograder.

Specific Uses:

- `getGradebook.m`
- `getRubric.m`
- `getFeedback.m`
- `getProblemFeedback.m`
- `runSubmission.m`

Description: General autograder information (especially hardcoded filenames) stored as a function that can be called by other functions for catered autograder use. Important: You can change the timeout settings [here](#)

Function: `getStudentIds.m`

Inputs: `filePath` - a string representing the path to student folders and the `grades.csv` file.

Outputs:

- `studentIds` - an Nx1 cell array of the student IDs.
- `studentFolders` - An Nx1 structure array representing the foldernames of student folders.

Specific Uses: `getGradebook.m`

Description: Given the correct file path, `getStudentIds` will extract all of the student IDs and folder names for these students. Both outputs are listed in the same order; that is, if student A's ID is at index N in the first output, his or her folder is at the same output in the second output.

Function: `gradeStudentSubmissions.m`

Inputs:

- `gradebook` – the gradebook object.
- `rubric` – the rubric object.
- `timeoutLogH` - a file handle representing the file used to keep track of timeouts.

Outputs: `gradebook` - a structure representing the updated gradebook.

Specific Uses: `runAutograder.m`

Description: Given the gradebook and the rubric, `gradeStudentSubmissions` will, using `runTestCase`, `run`, `grade`, and give feedback for that particular student in the gradebook. It returns the completely updated gradebook and is the starting and ending points for the student object.

Function: `gradeSubmission.m`

Inputs:

- `rubric` – the rubric object.
- `student` – the student object.

Outputs: `student` – the student object

Specific Uses: `gradeStudentSubmissions.m`

Description: Given the rubric and a specific student, `gradeSubmission` returns that same student with all of the points and, if appropriate, error messages needed to accurately describe the student's performance. This function does this on a student-by-student basis; that is, it only works on one student per call. It will additionally return the student's overall grade for this specific homework.

Function: `handleStudentTimeouts.m`

This function does not make sense and, frankly, I do not know why it is here. No function calls this in any discernable manner.

Function: `img2mat.m`

Inputs: `filename_img` - a string representing the name of the image file.

Outputs: `filename_mat` - a string representing the name of the .mat file.

Specific Uses: `convertSupportingFiles.m`

Description: This creates and returns a link to a .mat file that has the image data stored. This uses lossless conversion; all the original data is still there, just in a .mat format.

Function: `isFolderEmpty.m`

Inputs: `folderPath` - a string representing the path for the folder to check.

Outputs: `isEmpty` - a logical representing whether that folder was empty or not.

Specific Uses:

- `convertSupportingFiles.m`
- `runAutograder.m`

Description: This checks to see if the given file path is empty or not. Currently, it throws an error if the file path isn't a folder; this is considered a bug, not a feature, and will be fixed in the future.

Function: `isGradesCsvFormatDifferent.m`

Inputs: `gradebookTemplate` - a cell array that is the output of the `xlsread` for the csv file.

Outputs:

- `isFormatDifferent` - a logical that represents if the formatting has changed.
- `columnIndices` - the indices of our hard-coded columns.

Specific Uses: `getGradebook.m`

Description: This tells if the format of the CSV file that holds grades has changed at all. Regardless of whether it has or not, if the function can find the headers, the column indices will be produced faithfully. Note that if the format is different, the column indices cannot be trusted.

Function: `isVerbose.m`

Inputs:

Outputs: `is_verbose` - a logical true.

Specific Uses:

- `gradeStudentSubmissions.m`
- `uploadStudentFilesToServer.m`

Description: This should tell whether or not to print status updates to the console. It's broken now and only returns true.

Function: `lint.m`

Inputs: NONE

Outputs: NONE

Specific Uses: NONE

Description: `lint` uses MATLAB's built-in code checker to return possible problems. Anywhere there is an orange underline or red error in the autograder code files, `lint` will detail why this underlining occurred, where it occurred, and any details associated with it. It is a maintenance function.

Function: `loadjson.m`

Inputs:

- `fname` - a string representing the file path of the JSON file.
- `varargin*`: a structure representing options. Options are:
 - `SimplifyCell` (0 | 1): if 1, `loadjson` will use `cell2mat` on each element of the JSON data, and group arrays based on `cell2mat` rules.
 - `FastArrayParser` (0 | 1 | integer): if 0, use a legacy parser; if 1, use speed-optimized array parser. if greater than 1, the number represents the minimum dimension to enable the fast parser.
 - `ShowProgress` (0 | 1): if 1, show a progress bar.
 - `varargin` can be either a structure or set of name-value pairs.

Outputs: `dat` - a cell array representing the JSON data; `{}` are cell arrays, `[]` are normal arrays.

Specific Uses:

- `loadRubric.m`
- `lint.m`

Description: Given a JSON file path or JSON string, `loadjson` will return the data stored within. This was not created in house, and was instead downloaded from MathWork's FileExchange site.

Function: `loadRubric.m`

Inputs: `rubricJSONFilePath` - a string representing the file path to the `rubric.json` file.

Outputs: `rubric` – the rubric object.

Specific Uses: `getRubric.m`

Description: Given the JSON file for the solutions, `loadRubric` will generate the initial structure for the rubric. This is the starting point of the rubric object, though it is not the end point. See the rubric documentation for more information.

Function: `overrideBannedFunctions.m`

Inputs:

- `parentFolderPath` - a string representing the folder where the banned functions folders will be.
- `bannedFunctions` - a cell vector of all the banned functions. Can be a string IF it is only one banned function.
- `problemNumber` - a double representing the current homework problem number.
- `problemName` - a string representing the Function for the homework (i.e., the corresponding Function for the problem number).

Outputs: `bannedFunctionsFolderPath` - a string representing the path to the banned functions for a specific problem.

Specific Uses: `runSolutions.m`

Description: Given a set of banned functions and the immediate folder, `overrideBannedFunctions` will create empty `.m` files for each of these banned functions. Since MATLAB first looks in the current folder for a function definition, this effectively eliminates the specified functions. There is currently talk of adding `getMostRecentFiles.p` as a permanent banned function.

Function: `parseTestCase.m`

Inputs: `testCase` - a string representing the call for the specific test case.

Outputs:

- `inputVariables` - a cell array containing the input variables.
- `outputVariables` - a cell array containing the output variables.
- `variableInitialization` - a string that happens before the function call.
- `numberOfOutputs` - a double representing the number of outputs.

Specific Uses: `runSolutions.m`

Description: Given a function call as a string, `parseTestCase` will extract the right input and output variable names, as well as initialization information and the total number of outputs. Note that the first two cell array outputs are cell arrays of STRINGS, not the actual variable.

Function: `readGradesCsv.m`

Inputs: `gradesCsvFilePath` - a string that represents the `grades.csv` file path.

Outputs: `gradebookTemplate` - a cell array representation of the `grades.csv`

Specific Uses:

- `getGradebook.m`
- `writeGradesCsv.m`

Description: This returns the `grades.csv`, as a cell array. First the headers, a row of blanks, and the rest of the data. The cell array is the same size as the actual gradebook in terms of rows of data.

Function: `runAutograder.m`

Inputs:

- `homeworkZipFilePath*` - a string representing the path to the T-Square bulk download
- `rubricZipFilePath*` - a string representing the path to the hwrelease download from the Homework Team Drive.
- `destinationFolderPath*` - a string representing the path to your destination folder.

Outputs: NONE

Specific Uses: N/A

Description: This is the entry function for the autograder. All functions can eventually trace their caller to `runAutograder.m`. This function catches all errors and prints them back out onto the console, so that the runner may try to fix it.

Function: `runSolutions.m`

Inputs: rubric – the rubric object.

Outputs: rubric – the rubric object, updated

Specific Uses: `getRubric.m`

Description: Runs solution code and stores the outputs produced in the rubric structure

Function: `runSubmission.m`

Inputs:

- `rubric` - the rubric object.
- `student` - the student structure (a single student)
- `timeoutLogH` - a file handle to the timeout file.

Outputs: `student` - the updated student structure.

Specific Uses: `gradeStudentSubmissions.m`

Description: This runs the student's submission, and updates the student structure. This also writes to the timeout file. This takes care of any cheat-proofing and returns the student, grades and all.

Function: `runTestCase.m`

Inputs:

- `functionHandle` - a function handle to use with the inputs.
- `testCase` - a structure containing all the information about the test case.
- `inputs` - a structure containing the actual input arguments.
- `isSolution*` - a logical that represents if this test case is for a solution answer or not.
- `timeout*` - a double that represents how long to run the test case (a timeout value), in seconds.

Outputs: `output` - a structure containing the output information.

Specific Uses:

- `runSolutions.m`
- `runSubmission.m`

Description: `runTestCase.m` runs a specific test case, and returns all the necessary information about the test case. This is where the actual "work" happens in the autograder - this is where the test case is actually RUN and captured. The output will contain output variables, files, plots, and errors, with corresponding fields. Note that if you want to give a timeout, but not a solution value, you can simply say that `isSolution` is false or empty vector `([])`.

Function: unzipFile.m

Inputs:

- zipFilePath - a string representing the path to the .zip file.
- destinationFolderPath* - a string representing where to unzip the files to.

Outputs: folderPath - a string representing the unzipped folder path.

Specific Uses:

- getGradebook.m
- getRubric.m

Description: This unzips the files specified in the first argument into the folder specified by the second argument. If no second argument is given, the current working directory is assumed. This function produces identical results on any machine, but does use different methods on different Operating Systems. Specifically, if on a PC, `unzipFile` will use the 7-Zip command line command; if run on a UNIX machine, it will simply use MATLAB's built in `unzip` function. If there is an error while unzipping, `unzipFile` will throw an appropriate error.

Function: `uploadFile.m`

Inputs:

- `local_file_path` – a string representing the path of the local file to upload.
- `remote_file_path` – a string representing the path of the remote location.
- `channel*` – an input channel to use for uploading.
- `sftp_client*` – the sftp client to use for uploading.

Outputs: NONE

Specific Uses:

- `runAutograder.m`
- `uploadFilesToServer.m`
- `uploadStudentFilesToServer.m`

Description: Uploads file to `cs1371.gatech.edu` server.

Use Cases:

- `uploadFile h` or `uploadFile help` will print help information.
- `uploadFile v` or `uploadFile version` will print the version
- `paths = uploadFile('paths')` Returns the structure of the remote root paths
- `uploadFile(local_file_path, remote_file_path)` Uploads the local file to the remote path. If the remote path does not contain a file name, then the name of the local file will be used
- `uploadFile(local_file_path, remote_file_path, channel, sftp_client)` Uploads the local file to the remote path using the input channel and sftp_client. If the remote path does not contain a file name, then the name of the local file will be used.

Function: `uploadFilesToServer.m`

Inputs:

- `gradebook` - the gradebook object.
- `rubric` - the rubric object.

Outputs: NONE

Specific Uses: `runAutograder.m`

Description: This is experimental; it uploads all the student files to the MATLAB server, and the homework generator files as well. Right now, it takes way too long to complete, and is usually omitted from testing.

Function: `uploadHomeworkGeneratorFilesToServer.m`

Inputs:

- `rubric` - the rubric object.
- `homeworkNumber` - a double representing the current assignment
- `isResubmission*` - a logical representing if this is original or a resubmission.

Outputs: NONE

Specific Uses: `uploadFilesToServer.m`

Description: This uploads the homework generator files (.p) to the regrade server. if `isResubmission` is absent, it is assumed to be false. This is experimental!

Function: `uploadStudentFilesToServer.m`

Inputs: `gradebook` – the gradebook object.

Outputs: None

Specific Uses: `uploadFilesToServer.m`

Description: Uploads the structure containing the file paths of student submissions and feedback

Function: `visualizeValue.m`

Inputs: `value` - a variable value representing what needs to be visualized.

Outputs: `formattedValue` - a string that represents the HTML representation of the variable.

Specific Uses: `getProblemFeedback.m`

Description: Given a variable, `visualizeValue` will generate an HTML-ready visualization of said value. A notable exception is files - Files are not visualized explicitly, but instead via `visDiff` (that is, the file itself is not important; only how it compares to the solution file.)

Function: writeGradesCsv.m

Inputs:

- gradebook - the gradebook object.
- gradesCsv* - a cell array representing the content to write to grades.csv.

Outputs: NONE

Specific Uses: `runAutograder.m`

Description: This writes the grades to the grades.csv file provided in the bulk download from T-Square. gradesCsv is not required; if it is not given, it will read the grades from the original grades.csv to get the values to write. Regardless of whether or not the gradesCsv is given, if a gradebook is given, then the fifth column is overwritten with the grades from the gradebook. If the gradebook is empty, the original gradesCsv cell data is used instead. Note that gradebook is not optional; you MUST include a gradebook. However, this gradebook can be empty. This function is deprecated, however, and needs to be updated sometime in the near future.

Function: `xls2mat.m`

Inputs: `filename_xls` - a string representing the filepath of an excel document.

Outputs: `filename_mat` - a string representing the output filepath of the mat file

Specific Uses: `convertSupportingFiles.m`

Description: This converts an Excel workbook (.xls, .xlsx) into a .mat file for loading later.

Object: Rubric

The rubric is a structure that holds information about how to grade the test cases. It tells how much each test case is worth, what these test cases are, and what functions are assigned. It also has information about external files, and contains the actual function call, as a string.

Object: Gradebook

The gradebook is a structure that represents the grades from `grades.csv`. While the autograder is running, this structure is updated with grades.

Object: Student

Student is used for convenience; it is a transitory object (that is, it is created and consumed exclusively within the autograder). However, because it is standardized across the program, it is still considered to be a System-Wide object. In essence, it's a single entry from the *gradebook* object.