
Quickscope

Release 0.1.0

Nicholas Lambourne, Max Miller & Ella de Lore

Jul 07, 2021

CONTENTS:

- 1 Overview** **3**
- 1.1 Key Features 3
- 1.2 Build and Run 4
- 1.3 Documentation 5
- 1.4 Copyright & License 5

- 2 Indices and tables** **37**

- Python Module Index** **39**

- Index** **41**

OVERVIEW

Quickscope is used to build [Gradescope](#) autograders with [Chalkbox](#), the University of Queensland ITEE school's automatic marking system for programming assignments.

Quickscope is hosted at quickscope.uqcloud.net for public consumption.

1.1 Key Features

- Support for Python and Java based programming assessment.
- Extensible engine system for introducing new languages and environments.
- Dependency and environment management with [Poetry](#).
- A friendly web interface:



Create Autograder

Course Code
Course code of this assignment, e.g. CSSE2002

Assignment ID
Human readable identifier, e.g. ass1

Engine
ChalkBox engine to use when processing submission

Java Engine Configuration

Choose stages to run

<input type="checkbox"/> Conformance	Violation Penalty 1	Weighting 0
<input type="checkbox"/> Functionality		Weighting 0
<input type="checkbox"/> JUnit		Weighting 0
<input type="checkbox"/> Checkstyle	Violation Penalty 0.5	Weighting 0

Drop dependency libraries here...



1.2 Build and Run

Quickscope requires the Poetry dependency and environment management system as well as Python 3.8. If you don't have Poetry installed run:

```
pip install poetry
```

If you don't have Python 3.8 we'd suggest using the [pyenv](#) management system to install it:

```
pyenv install 3.8.2
```

Once you have those, you can install Quickscope:

```
git clone https://github.com/UQTools/quickscope
cd quickscope
poetry shell
poetry install
poetry run build
```

When the generation of the front end is complete you can run the Quickscope server:

```
poetry run flask run
```

You should now be able to access the service through your browser at `0.0.0.0:5000`.

1.3 Documentation

Documentation for the project is available at quickscope.readthedocs.org. This includes guides on how to use the tool and adapt it to your needs.

1.4 Copyright & License

Quickscope is copyright Nicholas Lambourne, Max Miller & Ella de Lore.

The Quickscope logo is copyright Nicholas Lambourne.

This tool has been licensed for general use under a permissive MIT license available [here](#).

1.4.1 Guides

Guides for students, tutors and course coordinators using Quickscope.

Creating a JavaEngine Autograder

This guide explains the process of configuring and generating an autograder bundle in Quickscope, using the Java Engine for ChalkBox.

Generic Setup

1. Visit <http://quickscope.uqcloud.net> and log in with your UQ credentials.
2. Choose a course code and assignment identifier for the autograder. These will be displayed in the instructor-facing output for debugging purposes each time the autograder runs.
3. Choose the JavaEngine under the 'Engine' dropdown box. A set of engine-specific options will appear below.

Quickscope

Create Autograder

Course Code

CSSE2002

Course code of this assignment, e.g. CSSE2002

Assignment ID

ass1

Human readable identifier, e.g. ass1

Engine

JavaEngine

ChalkBox engine to use when processing submission

Java Engine Configuration

Here, you can select the individual stages to be run as part of the autograder. All stages are optional and can be run separately or together with one or more other stages. Selecting the checkbox next to the name of a stage will enable it, and any stage-specific options will appear below.

Each stage has a weighting, which can be changed by entering a number in the associated text field. This represents the total number of marks, out of 100, allocated to this stage. A submission that scores perfectly in a given stage will receive the number of marks entered here. If a weighting is 0, the stage will still be run, however no marks will be allocated to it.

Immediately below the options for each stage are file drop zones for dependency libraries and the correct solution directory, explained below.

Java Engine Configuration

Choose stages to run

<input type="checkbox"/> Conformance	Violation Penalty 1	Weighting 0
<input type="checkbox"/> Functionality		Weighting 0
<input type="checkbox"/> JUnit		Weighting 0
<input type="checkbox"/> Checkstyle	Violation Penalty 0.5	Weighting 0

Dependencies

Any dependency JARs that are required by the assignment should be added here, for example, JUnit and Hamcrest. If the Checkstyle stage is enabled, the Checkstyle JAR should be added here as well.

Drop dependency libraries here...



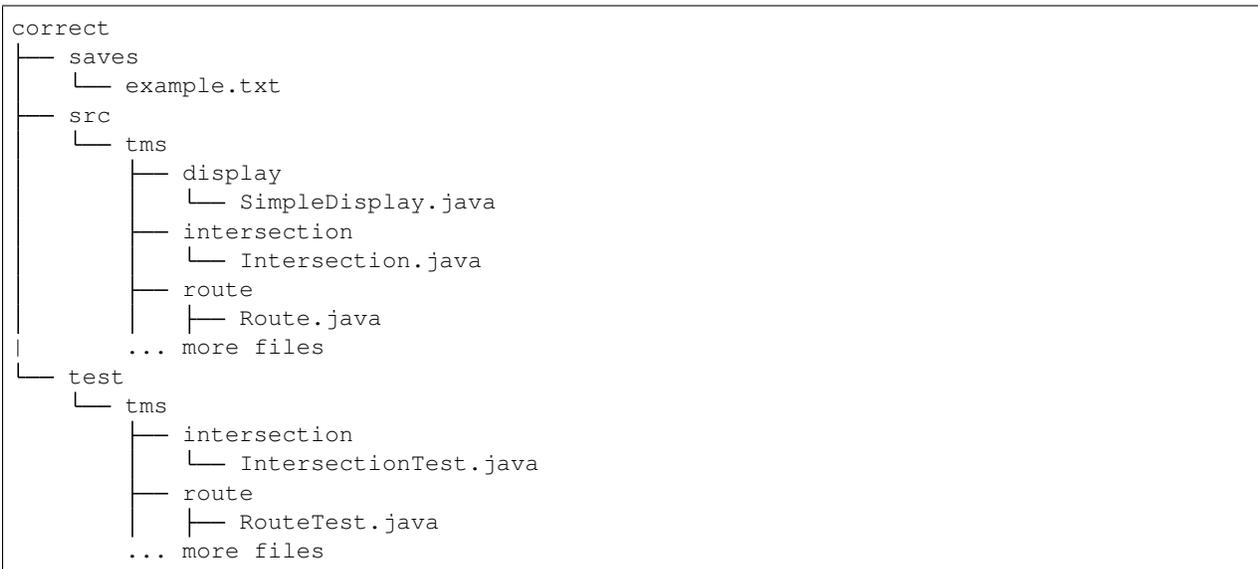
checkstyle-8.36-all.jar ✕

hamcrest-core-1.3.jar ✕

junit-4.12.jar ✕

Correct Solution

The directory containing the assignment solution should be dragged here. This directory should contain the two directories `src` and `test`, along with any extra text files provided by course staff for the purposes of testing I/O-related classes. For example, if the directory to be added to Quickscope is called `correct`, the structure should be:



The correct solution directory should contain all the solution code for the assignment inside `src/`, as well as all the tests to be run against the student submission (if the functionality stage is enabled) in `test/`.

Test Visibility

To provide immediate feedback to students on submission, a subset of the uploaded JUnit tests in the `test/` directory can be marked as visible at the time of student submission, rather than the default of only after the grades have been published/finalised for the assignment. Immediately visible tests will appear in the results section when a student makes a submission, along with the mark received for that test and the test output if it failed.

To mark an individual JUnit test as visible, use the `@Deprecated` annotation. For example:

```
@Test
@Deprecated
public void toStringTest() {
    assertEquals("A", intersection1.toString());
}
```

Test Weightings

Weightings can be used to allocate more marks to a particular unit test where it is deemed necessary.

The weighting of each individual JUnit test can be modified by adding the `timeout` parameter to the `@Test` annotation. Weightings are integers between one and nine inclusive, representing a weighting of one and nine times the regular test weighting respectively.

When including the `timeout` parameter of a test to specify a weighting, it is recommended to add a sufficiently large value such that the test does not actually time out for the given weighting, such as one million milliseconds.

For example, to give a test a weighting of three times the regular weighting:

```
@Test(timeout = 1000000 + 3)
public void hashCodeTest() {
    assertEquals(a.hashCode(), b.hashCode());
}
```

Note that the recommended approach to adding timeouts to tests is to use a class-wide timeout rule as shown below.

```
public class MyTest {
    @Rule
    public Timeout timeout = Timeout.seconds(1);

    ...
}
```

Conformance

After enabling the conformance stage by clicking the checkbox under Java Engine Configuration, the violation penalty can be changed by entering a number in the input field. This number represents the number of marks deducted for each instance of non-conformance to the expected public API found in the correct solution. The total of all deductions is capped at the weighting assigned to the stage, meaning students cannot receive a negative mark.

A file drop zone is provided for specifying the expected file structure of a student submission. This should generally be similar to the correct solution directory, however, only the tests classes that are assessable should be present. This is to avoid extra test classes being marked as 'missing' in the conformance check.

For example, if the directory to be added to Quickscope is called `correct_structure` and the assessable test classes are `IntersectionTest` and `DemoPressurePadTest`, the structure should be:

```
correct_structure
├── saves
│   └── example.txt
├── src
│   └── tms
│       ├── display
│       │   └── SimpleDisplay.java
│       ├── intersection
│       │   └── Intersection.java
│       ├── route
│       │   └── Route.java
│       └── ... more source files
└── test
    └── tms
        └── intersection
            └── IntersectionTest.java
```

(continues on next page)

(continued from previous page)



JUnit

The JUnit classes that are assessable in the assignment should be selected by clicking on the rounded boxes representing files under the “Preview:” section below the correct solution drop zone.

Once a file has been selected, it will appear in the list of Assessable Test Classes. To remove a file from the list, simply click its rounded box again.

JUnit

Assessable Test Classes

Select classes by clicking on files above

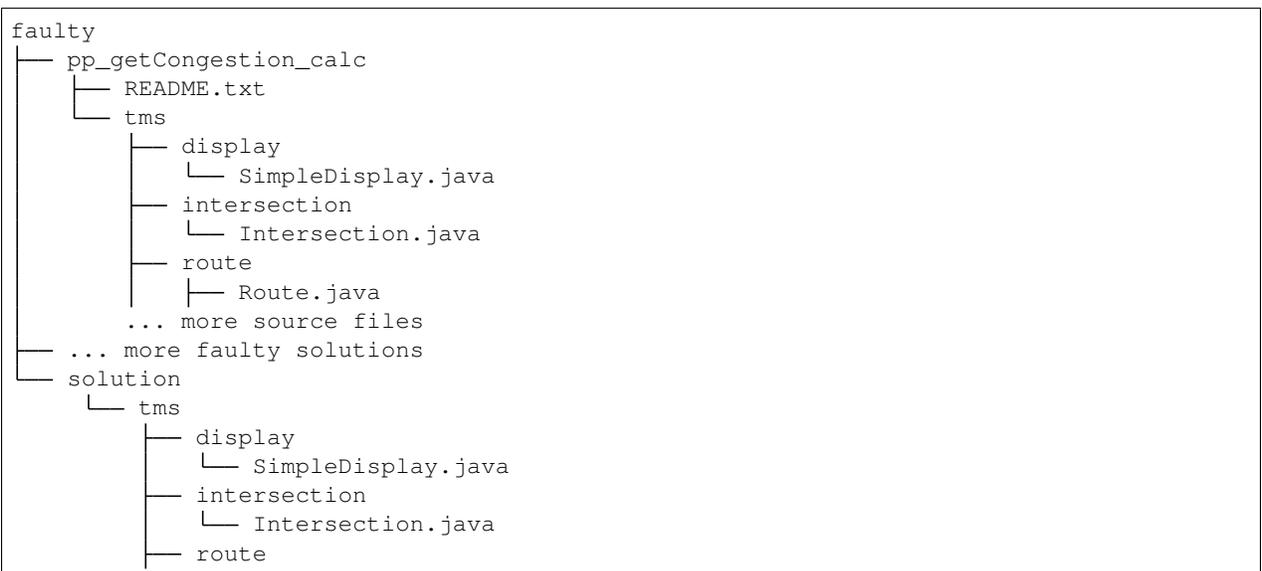
 IntersectionTest.java

 DemoPressurePadTest.java

Below the list of assessable test classes, there is a file drop zone for uploading a directory containing a set of faulty solutions to the assignment, used when assessing submitted JUnit tests. The correct implementation of the assignment should also be included in this directory, under `solution/`.

As a suggestion, each subdirectory should be named according to the test class it is assessing. For example, `pp` for `DemoPressurePadTest`. These subdirectories should act as the `src/` directory for each solution, with packages as immediate subdirectories.

For example, if the directory to be added to Quickscope is called `faulty`, the structure should be:



(continues on next page)

(continued from previous page)

```
|  └─ Route.java  
... more source files
```

Checkstyle

After enabling the Checkstyle stage by clicking the checkbox under Java Engine Configuration, the violation penalty can be changed by entering a number in the input field. This number represents the number of marks deducted for each style violation detected by the Checkstyle tool when run on the `src/` directory of the submitted assignment.

A text input field is provided for specifying paths that should be excluded from being checked for style violations by the Checkstyle tool. Excluded paths can be directories (using a trailing slash) or individual files. If a directory is specified, the entire directory will be ignored when checking for style violations. This functionality can be useful when the assignment contains code provided by course staff that is not compliant with the style guide, for example, a package containing GUI-related code.

To add an excluded path, enter the path in the input field, ensuring the path is relative to the root directory of the submission, `/autograder/submission/`. Then, click the “+” button inside the input field. The path will display under “Excluded Paths”.

A file drop zone is provided to specify the configuration file to be used by the Checkstyle tool. This is an `.xml` file containing a list of all violation types to detect, and any associated options.

Checkstyle

Excluded Paths

 /autograder/submission/src/tms/display/SimpleDisplay.java

Add excluded path



Relative to submission directory, e.g. src/tms/display/

Drop Checkstyle configuration file here...



Generating and Uploading Autograder

Finally, once all stages have been configured, the autograder is ready to be generated. Simply click the “Generate Autograder Bundle” button at the bottom of the page, and a `.zip` file containing the entire autograder suite will be downloaded automatically. This file can then be uploaded to Gradescope.

Creating a PythonEngine Autograder

This guide explains the process of configuring and generating an autograder bundle in Quickscope, using the Python Engine for ChalkBox.

Generic Setup

1. Visit <http://quickscope.uqcloud.net> and log in with your UQ credentials.
2. Choose a course code and assignment identifier for the autograder. These will be displayed in the instructor-facing output for debugging purposes each time the autograder runs.
3. Choose the PythonEngine under the 'Engine' dropdown box. A set of engine-specific options will appear below.

Create Autograder

Course Code

Course code of this assignment, e.g. CSSE2002

Assignment ID

Human readable identifier, e.g. ass1

Engine

ChalkBox engine to use when processing submission

Python Engine Configuration

Here you will be required to provide the needed files for the engine.

You will first have to provide the expected name of the student submitted file (e.g a1) and the name of test script (e.g test_a1.py).

Python Engine Configuration

File Name

a1

Name of submitted file (without extension, e.g. a1)

Test Runner Name

test_a1.py

(Just the) Name of the test file (with extension, e.g. test_a1.py). This should be included in the `included` directory.

Drop included directory here...



Included Folder

You will need to provide a folder of included files to Quickscope containing:

- The test script and dependencies (e.g. for CSSE1001, `test_a1.py` and `testrunner.py`)
- Support files for testing (e.g. config files)
- Support files for student code (any files that have been provided to students as support code)

This is an example of an included folder for CSSE1001, using the 2019 assignment 1:

```
included
├── test_data
│   ├── main_fixed_win.in
│   ├── main_fixed_win.out
│   └── ... more files
├── a1_test.py
├── testrunner.py
├── a1_support.py
├── WORDS_ARBITRARY.txt
└── WORDS_FIXED.txt
```

Visible Tests

You will need to provide a text file that contains the names of test classes that will be available to students before the due date. The results from the tests named in the file will be visible to students each time they upload a submission to Gradescope.

JSON Formatter

You will need to provide a Python script that reformats the test runner output to be in a format accepted by Gradescope. A JSON Formatter for the current CSSE1001 Test Runner can be found [on GitHub](#).

Generating and Uploading Autograder

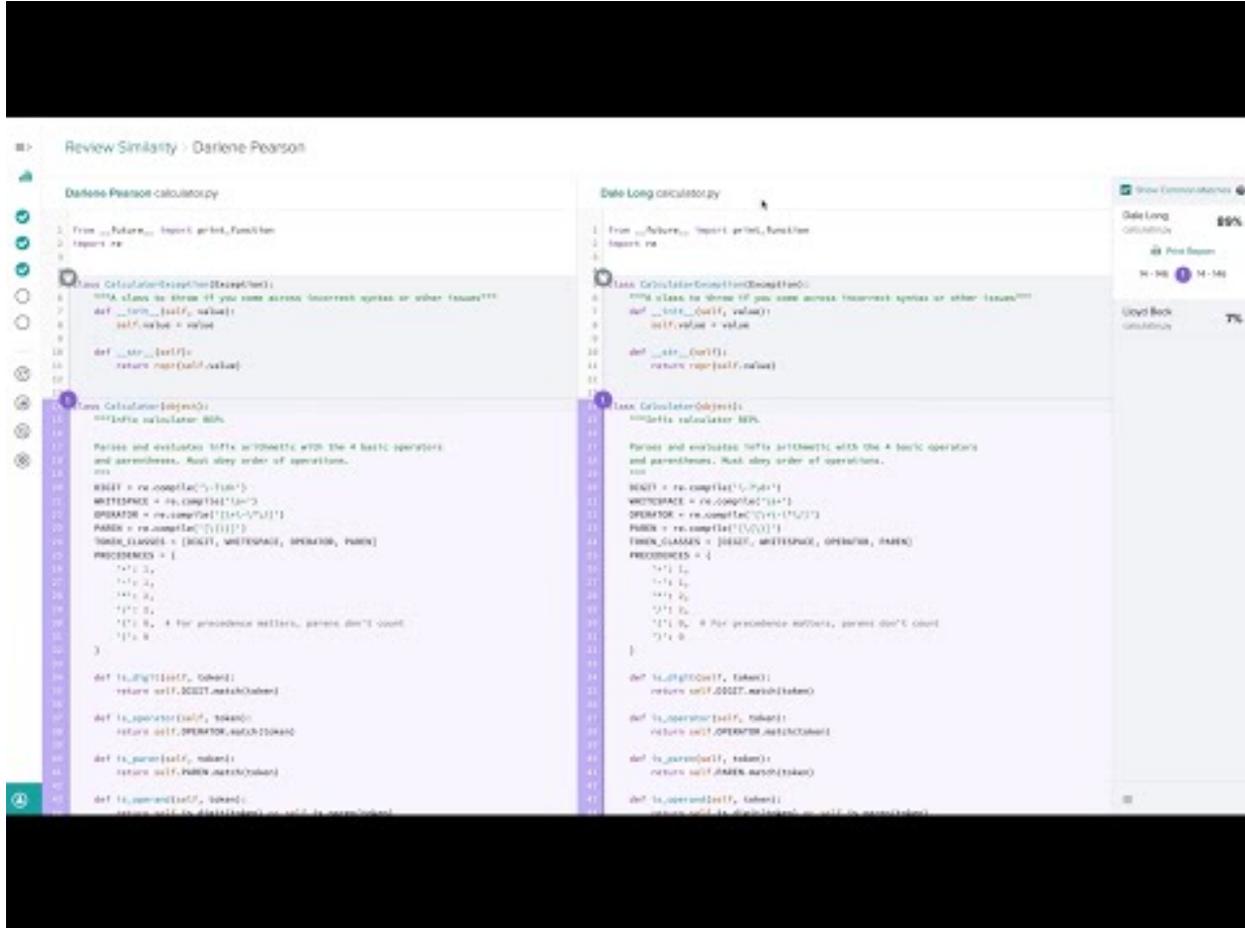
Finally, once all stages have been configured, the autograder is ready to be generated. Simply click the “Generate Autograder Bundle” button at the bottom of the page, and a `.zip` file containing the entire autograder suite will be downloaded automatically. This file can then be uploaded to Gradescope.

Setting up an Assignment in Gradescope

This guide explains the process of creating a programming assignment in Gradescope, using an autograder generated by Quickscope.

Creating the Assignment

Firstly, follow the steps outlined in Gradescope’s demonstration video below to create a programming assignment using the Gradescope interface.



Assignment Settings

The following options should be set in the “Assignment Settings” panel when creating the assignment:

- Autograder points should be set to the total of the weights of all the automatically marked components of the assignment. This number should be the maximum possible number of marks achievable in the autograder.
 - For JavaEngine, this will be the sum of all the weights assigned to the JavaEngine stages.
 - For PythonEngine, this will be the maximum achievable functionality score, excluding the manually marked style component.
- “Enable Manual Grading” should be enabled if there is to be a manually marked style component
- “Due Date” should be set to the official due date of the assignment. Submissions made after this date but before the Late Due Date will be marked as “Late” when marking, along with the number of days it is late.
- “Allow late submissions” should be enabled.
- “Late Due Date” should be set to the last expected submission date of the longest extension that is allowed for the assignment. Students will not be allowed to submit after this date. Course staff can still submit on behalf of

students, though students cannot see their autograder results on submission.

Outline

In the “Outline” section, an additional question should be created for the manual style component, including the weighting assigned to manual style.

Outline for **Test Assignment**

100 points total

Create questions and subquestions via the + buttons below. Reorder and indent questions by dragging them in the outline.

#	TITLE	POINTS
1	Autograder	85
2	Manual Style	15
+ new question		

Save Outline

Cancel

Autograder

Finally, the autograder `.zip` generated by Quickscope should be uploaded on this page by selecting “Zip file upload”, “Select Autograder (.zip)” then “Update Autograder”.

After uploading the autograder, you can test that it works correctly by clicking “Test Autograder” and uploading a student submission, for example, the correct solution for the assignment written by course staff.

Configure Autograder

Upload your autograder code and change settings here. You can also come back to this step later, but submissions will not be automatically graded until then. Please follow our [guidelines](#) for structuring your autograder.

Note: Uploading an autograder zip file will automatically update your Dockerhub image name once it is built successfully.

AUTOGRADER CONFIGURATION

Zip file upload Manual Docker Configuration

AUTOGRADER

 Please select a file

Select Autograder (.zip)

Update Autograder

 Test Autograder

Other Settings

By clicking the “Settings” link at the bottom of the left sidebar after entering the assignment page, other various settings can be enabled that are not present in the “Assignment Settings” panel shown when creating the assignment. Key changes that can be made here include:

- Specifying which submission methods are allowed. Submission via direct file upload, GitHub and Bitbucket can be enabled or disabled.
- Specifying ignored files. Files that match these conditions will be filtered out from a student’s submission before being processed with ChalkBox. This may be useful if ChalkBox is having problems with certain types of files.
- Changing the autograder container’s system specifications. More CPU and RAM may improve performance.
- Changing the autograder timeout. The default timeout is 10 minutes before an autograder instance is killed, however this can be increased up to 40 minutes.

SUBMISSION METHODS ENABLED

- Upload
- GitHub
- Bitbucket

IGNORED FILES

```
__MACOSX/  
.DS_Store
```

Follows [gitignore](#) format.

Autograder Settings

CONTAINER SPECIFICATIONS

Your autograder will have access to at least the portion of CPU allocated, and at most the memory allocated

- 0.5 CPU, 0.75GB RAM
- 1.0 CPU, 1.5GB RAM
- 2.0 CPU, 3.0GB RAM
- 4.0 CPU, 6.0GB RAM

AUTOGRADER TIMEOUT

Your autograder will be timed out after this amount of time.

10 minutes ▾

Save

🗑 Delete assignment

Style Marking in Gradescope

This guide covers the process of style marking student assignments in Gradescope.

Navigating to Marking Screen

In each course, assignments can be found under the `Assignments` tab.

This shows all the current assignments for the course.

gradescope <≡

CSSE1001/7030
Introduction to Software Engineering

- Dashboard
- Assignments**
- Roster
- Extensions
- Course Settings

INSTRUCTORS

- Maxwell
- Ella de Lore
- Nicholas Lambourne

1 Assignment

NAME	POINTS	RELEASED	DUE (AEST)	SUBMISSIONS	% GRADED	PUBLISHED	REGRADES
a1	43.0	SEP 30	OCT 08 AT 1:00PM	1	100%	ON	

To mark a specific assignment, click on its name in the list.

Then select `Grade Submissions` in the left-hand menu to navigate to the questions in the assignment.

For this assignment, style marking is graded under one question, in this case, “Manual style”.

gradescope <≡

< Back to CSSE1001/7030

a1

- Grade Submissions**
- Edit Outline
- Configure Autograder
- Manage Submissions
- Review Grades

Regrade Requests

Statistics

Review Similarity

Settings

Grading Dashboard

QUESTION	POINTS	PROGRESS	GRADED BY
2: Manual Style	10.0	100%	EDL, EDL

Selecting the name of the question will take you to a list of the names of students.

Marking

The Gradescope marking interface uses a number of shortcuts to assist with marking. Please watch the following videos to learn about the interface and these shortcuts.

The screenshot displays the Gradescope marking interface. The main area shows a question titled "Q3. Chemistry" with the sub-question "Q3.1 [3pt] Draw the Lewis structure for CO₂". The student's handwritten answer is "O-C-O". Below this, question "Q3.2 [1pt] Which of the following are noble gases?" is shown with radio button options for "Argon" (unchecked) and "Helium" (checked). The right-hand panel, titled "3.1 Lewis Structure", shows the question's score of "1.0 / 3.0 pts" and a list of marking points: "0.0" for "Correct", "1.0" for "Missing electrons around the O's", and "2.0" for "Single bond". The bottom status bar indicates "Submissions: 2 of 10" and includes buttons for "Previous question", "Previous", "Next", and "Next question".

The screenshot displays the Quickscope interface for a chemistry problem. The main area is divided into two sections. The top section, titled "Q3. Chemistry", contains the question "Q3.1 [1pt] Draw the Lewis structure for CO₂". Below the question is a hand-drawn Lewis structure of carbon dioxide, showing a central carbon atom double-bonded to two oxygen atoms, with lone pairs on the oxygen atoms. The bottom section, titled "Q3.2 [2pt] Which of the following are noble gases?", lists four options: Argon, Helium, Oxygen, and Hydrogen. Each option has a radio button next to it, with Argon and Helium selected. On the right side of the interface, there is a sidebar for "21 Lewis Structure" with a score of "3.0 / 3.0 pts". The sidebar includes a "Correct" status, a "Wrong answers around the Quickscope world" section, and a "Single bond" section. At the bottom of the interface, there is a teal bar with "Submissions: 8 of 19" and buttons for "Unsubmitted", "Marked", "Used In", and "Reset (Updated 3)".

Commenting

Comments can be added to student code by clicking on a section of student code. A comment box will then appear where you can add your comment and then save.

The screenshot shows the 'Autograder Results' page. The main area displays the source code for 'src/tms/JDKTest.java'. The code is as follows:

```

1 package tms;
2
3 public class JDKTest {
4
5     public void version () {
6
7     }
8
9 }
10

```

Below the code, there are several other files listed with 'Download' buttons: 'src/tms/sensors/DemoPressurePad.java', 'src/tms/sensors/DemoSpeedCamera.java', 'src/tms/sensors/SpeedCamera.java', and 'src/tms/sensors/DemoSensor.java'. A comment box is open above the code with the text 'Enter your comment...' and 'Save' and 'Cancel' buttons.

On the right side, the rubric is visible. It shows '2: Manual Style' and '0 OF 2 GRADED'. The total points are '15.0 / 15.0 pts'. There are also buttons for 'Add Rubric Item', 'Create Group', and 'Import...'. The 'SUBMISSION SPECIFIC ADJUSTMENTS' section shows 'Point Adjustment 0' and a text box for comments. The 'APPLY PREVIOUSLY USED COMMENTS' section has a dropdown menu.

The student will be able to view comments linked to the code that you clicked on.

Shortcuts

- 1 - 9 - Select Rubric Group
- QWERTY - Select Rubric Item
- Left Arrow - Previous Student
- Right Arrow - Next Student
- z - Next Ungraded
- a - Show all submissions
- . - Next Question
- , - Previous Question

Understanding Gradescope Feedback CSSE2002

When you have submitted your assignment to Gradescope, you will receive feedback on your work. If you submit before the due date you will receive some “pre-checks” on your assignment. Full feedback will be available at the time of grade release.

Before Due Date

Before the due date of the assignment, submitting your assignment to Gradescope will allow you to receive “pre-checks” on your submitted code.

The screenshot shows the 'Autograder Results' interface. At the top, there are tabs for 'Results' and 'Code'. The main content is divided into several sections:

- Compilation:** A box indicating 'Submission successfully compiled'.
- Conformance (10.0/10.0):** A section with 'Missing files:' and 'Extra files:' labels. Below, a list of classes is shown, each followed by 'conforms'. The classes include:
 - tms.display.SimpleDisplay
 - tms.intersection.Intersection
 - tms.route.Route
 - tms.route.SpeedSign
 - tms.route.TrafficLight
 - tms.route.TrafficSignal
 - tms.sensors.DemoPressurePad
 - tms.sensors.DemoSensor
 - tms.sensors.DemoSpeedCamera
 - tms.sensors.PressurePad
 - tms.sensors.Sensor
 - tms.sensors.SpeedCamera
 - tms.util.DuplicateSensorException
 - tms.util.RouteNotFoundException
 - tms.util.TimedItem
 - tms.util.TimedItemManager
- IntersectionTest.toStringTest (0.5633802816901409/0.5633802816901409):** A box showing a single test result.
- RouteTest.testToStringBasic (0.5633802816901409/0.5633802816901409):** A box showing another test result.

On the right side, a sidebar provides summary information:

- STUDENT:** Ella Test
- AUTOGRADER SCORE:** - / 85.0
- FAILED TESTS:** Automated Style (2.5/15.0)
- PASSED TESTS:**
 - Conformance (10.0/10.0)
 - IntersectionTest.toStringTest (0.5633802816901409/0.5633802816901409)
 - RouteTest.testToStringBasic (0.5633802816901409/0.5633802816901409)
 - RouteTest.testToStringSpeedSign (0.5633802816901409/0.5633802816901409)
 - DemoPressurePadTest.toStringTest (0.5633802816901409/0.5633802816901409)
 - DemoSpeedCameraTest.toStringTest (0.5633802816901409/0.5633802816901409)
- QUESTION 2:** Manual Style - / 15.0 pts

At the bottom, there are three buttons: 'Submission History', 'Download Submission', and 'Resubmit'.

These checks are:

- **Compilation** - checks if your program compiles, if not, give the compilation error output explaining why your code failed to compile.

This screenshot shows the 'Compilation' section of the autograder results. It features a header 'Compilation' and a message box stating 'Submission successfully compiled'.

- **Conformance** - checks if your program conforms to the provided specification.

This screenshot shows the 'Conformance (10.0/10.0)' section. It lists 'Missing files:' and 'Extra files:' which are both empty. Below, under 'Class conformance:', a list of classes is shown, each followed by 'conforms'. The classes are:

- tms.display.SimpleDisplay
- tms.intersection.Intersection
- tms.route.Route
- tms.route.SpeedSign
- tms.route.TrafficLight
- tms.route.TrafficSignal
- tms.sensors.DemoPressurePad
- tms.sensors.DemoSensor
- tms.sensors.DemoSpeedCamera
- tms.sensors.PressurePad
- tms.sensors.Sensor
- tms.sensors.SpeedCamera
- tms.util.DuplicateSensorException
- tms.util.RouteNotFoundException
- tms.util.TimedItem
- tms.util.TimedItemManager

- **Functionality** - a small number of selected functionality tests may be provided for you to see the functionality of your program. These tests are a subset of the full suite of tests used to mark your assignment’s functionality.

IntersectionTest.toStringTest (0.5633802816901409/0.5633802816901409)

RouteTest.testToStringBasic (0.5633802816901409/0.5633802816901409)

RouteTest.testToStringSpeedSign (0.5633802816901409/0.5633802816901409)

DemoPressurePadTest.toStringTest (0.5633802816901409/0.5633802816901409)

DemoSpeedCameraTest.toStringTest (0.5633802816901409/0.5633802816901409)

- JUnit Test Compilation - checks if your provided JUnit tests compile with our solution. If your tests do not compile with our solution then you will receive 0 marks for the JUnit section of the assignment

JUnit compilation

```
Output:
JUnit test file tms/intersection/IntersectionTest.java found
JUnit test file tms/intersection/IntersectionTest.java compiles

JUnit test file tms/sensors/DemoPressurePadTest.java found
JUnit test file tms/sensors/DemoPressurePadTest.java compiles
```

- Style - automatically checks your program for style violations. These violations will be deducted from your total style marks.

Automated Style (2.5/15.0)

```
Starting audit...
[WARN] /src/tms/intersection/Intersection.java:17: First sentence of Javadoc is missing an ending period. [SummaryJavadoc]
[WARN] /src/tms/intersection/Intersection.java:19: First sentence of Javadoc is missing an ending period. [SummaryJavadoc]
[WARN] /src/tms/intersection/Intersection.java:69:48: WhitespaceAround: '{' is not preceded with whitespace. [WhitespaceAround]
[WARN] /src/tms/sensors/DemoSensor.java:16: First sentence of Javadoc is missing an ending period. [SummaryJavadoc]
[WARN] /src/tms/sensors/DemoSensor.java:18: First sentence of Javadoc is missing an ending period. [SummaryJavadoc]
[WARN] /src/tms/sensors/DemoSensor.java:20: First sentence of Javadoc is missing an ending period. [SummaryJavadoc]
[WARN] /src/tms/sensors/DemoSensor.java:22: First sentence of Javadoc is missing an ending period. [SummaryJavadoc]
[WARN] /src/tms/util/TimedItemManager.java:27:5: Missing a Javadoc comment. [JavadocVariable]
[WARN] /src/tms/util/TimedItemManager.java:28:5: Missing a Javadoc comment. [JavadocVariable]
[WARN] /src/tms/util/TimedItemManager.java:51: Line is longer than 80 characters (found 83). [LineLength]
[WARN] /src/tms/util/TimedItemManager.java:55:56: '(' is preceded with whitespace. [MethodParamPad]
[WARN] /src/tms/route/TrafficLight.java:9: First sentence of Javadoc is missing an ending period. [SummaryJavadoc]
[WARN] /src/tms/route/TrafficSignal.java:12:5: Missing a Javadoc comment. [JavadocVariable]
[WARN] /src/tms/route/TrafficSignal.java:13:5: Missing a Javadoc comment. [JavadocVariable]
[WARN] /src/tms/route/TrafficSignal.java:14:5: Missing a Javadoc comment. [JavadocVariable]
[WARN] /src/tms/route/TrafficSignal.java:15:5: Missing a Javadoc comment. [JavadocVariable]
[WARN] /src/tms/route/Route.java:21: First sentence of Javadoc is missing an ending period. [SummaryJavadoc]
[WARN] /src/tms/route/Route.java:23: First sentence of Javadoc is missing an ending period. [SummaryJavadoc]
[WARN] /src/tms/route/Route.java:25: First sentence of Javadoc is missing an ending period. [SummaryJavadoc]
[WARN] /src/tms/route/Route.java:25: Line is longer than 80 characters (found 82). [LineLength]
[WARN] /src/tms/route/Route.java:27: First sentence of Javadoc is missing an ending period. [SummaryJavadoc]
[WARN] /src/tms/route/Route.java:29: First sentence of Javadoc is missing an ending period. [SummaryJavadoc]
[WARN] /src/tms/route/Route.java:31: First sentence of Javadoc is missing an ending period. [SummaryJavadoc]
[WARN] /src/tms/route/Route.java:184:56: ')' is preceded with whitespace. [ParenPad]
[WARN] /src/tms/route/Route.java:216: Line is longer than 80 characters (found 87). [LineLength]
Audit done.
```

Assignment Feedback

When the assignment grades are published you will be able to view you final grades and feedback. The screen will be similar to the feedback explained above, except there will be three main differences.

Autograder Results

Results
Code

Compilation

Submission successfully compiled

Conformance (10.0/10.0)

Missing files:

Extra files:

Class conformance:

```
tms.display.SimpleDisplay conforms
tms.intersection.Intersection conforms
tms.route.Route conforms
tms.route.SpeedSign conforms
tms.route.TrafficLight conforms
tms.route.TrafficSignal conforms
tms.sensors.DemoPressurePad conforms
tms.sensors.DemoSensor conforms
tms.sensors.DemoSpeedCamera conforms
tms.sensors.PressurePad conforms
tms.sensors.Sensor conforms
tms.sensors.SpeedCamera conforms
tms.util.DuplicateSensorException conforms
tms.util.RouteNotFoundException conforms
tms.util.TimedItem conforms
tms.util.TimedItemManager conforms
```

IntersectionTest.addConnectionConnectionDuplicateDifferentSpeedsTest (0.5633802816901409/0.5633802816901409)

IntersectionTest.addConnectionConnectionDuplicateTest (0.5633802816901409/0.5633802816901409)

STUDENT
Ella Test

AUTOGRADER SCORE
72.5 / 85.0

FAILED TESTS
Automated Style (2.5/15.0)

PASSED TESTS
Conformance (10.0/10.0)

```
IntersectionTest.addConnectionConnectionDuplicate
(0.5633802816901409/0.5633802816901409)
IntersectionTest.addConnectionConnectionDuplicate
(0.5633802816901409/0.5633802816901409)
IntersectionTest.addConnectionNegativeSpeedTest
(0.5633802816901409/0.5633802816901409)
IntersectionTest.addConnectionTest
(0.5633802816901409/0.5633802816901409)
IntersectionTest.addConnectionZeroSpeedTest
(0.5633802816901409/0.5633802816901409)
IntersectionTest.getConnectedIntersectionsEmptyLit
(0.5633802816901409/0.5633802816901409)
IntersectionTest.getConnectedIntersectionsTest
(0.5633802816901409/0.5633802816901409)
IntersectionTest.getConnectionDoesNotExistsTest
(0.5633802816901409/0.5633802816901409)
IntersectionTest.getConnectionExistsTest
(0.5633802816901409/0.5633802816901409)
```

- Full Functionality - You will be able to see the results from all the functionality tests for the assignment and whether you passed or failed each test.
- JUnit Results - You will be able to see the results from your tests. Your tests are run against the solution to the assignment and a number of faulty solutions. Your mark is determined by the number of faulty solutions that pass **fewer** of your tests than are passed by the actual solution. The name in parentheses tells you which of your test classes that faulty solution was accessing, along with a brief identifier for the faulty solution.

JUnit (ds_currentValue) (1.0/1.0)

Number of your tests that failed when run against this implementation: 4
 Tests failed for this implementation:
 getCongestionInitialTest(tms.sensors.DemoPressurePadTest): expected:<0> but was:<25>
 countTrafficNoTimeTest(tms.sensors.DemoPressurePadTest): expected:<1> but was:<0>
 getCongestionRoundingTest(tms.sensors.DemoPressurePadTest): expected:<56> but was:<11>
 getCongestionBasicTest(tms.sensors.DemoPressurePadTest): expected:<0> but was:<25>

JUnit (ds_getThreshold) (1.0/1.0)

Number of your tests that failed when run against this implementation: 3
 Tests failed for this implementation:
 getThreshold(tms.sensors.DemoPressurePadTest): expected:<4> but was:<5>
 getCongestionRoundingTest(tms.sensors.DemoPressurePadTest): expected:<56> but was:<55>
 getCongestionBasicTest(tms.sensors.DemoPressurePadTest): expected:<25> but was:<20>

JUnit (ds_oneSecond) (1.0/1.0)

Number of your tests that failed when run against this implementation: 4
 Tests failed for this implementation:
 countTrafficArrayLoopTest(tms.sensors.DemoPressurePadTest): Index -1 out of bounds for length 7
 countTrafficSomeTimeTest(tms.sensors.DemoPressurePadTest): expected:<2> but was:<3>
 getCongestionRoundingTest(tms.sensors.DemoPressurePadTest): expected:<11> but was:<56>
 getCongestionBasicTest(tms.sensors.DemoPressurePadTest): expected:<25> but was:<0>

- Style Feedback - You can view your full style feedback by navigating to the “Code” tab on the results page.

Autograder Results

Results Code

Compilation
Submission successfully compiled

Conformance (10.0/10.0)

Missing files:
Extra files:

```

Class conformance:
tms.display.SimpleDisplay conforms
tms.intersection.Intersection conforms
tms.route.Route conforms
tms.route.SpeedSign conforms
tms.route.TrafficLight conforms
tms.route.TrafficSignal conforms
tms.sensors.DemoPressurePad conforms
tms.sensors.DemoSensor conforms
tms.sensors.DemoSpeedCamera conforms
tms.sensors.PressurePad conforms
tms.sensors.Sensor conforms
tms.sensors.SpeedCamera conforms
tms.util.DuplicateSensorException conforms
tms.util.RouteNotFoundException conforms
tms.util.TimedItem conforms
tms.util.TimedItemManager conforms
    
```

IntersectionTest.addConnectionConnectionDuplicateDifferentSpeedsTest (0.5633802816901409/0.5633802816901409)

IntersectionTest.addConnectionConnectionDuplicateTest (0.5633802816901409/0.5633802816901409)

STUDENT
Ella Test

AUTOGRADER SCORE
72.5 / 85.0

FAILED TESTS
Automated Style (2.5/15.0)

PASSED TESTS
Conformance (10.0/10.0)
IntersectionTest.addConnectionConnectionDuplicate (0.5633802816901409/0.5633802816901409)
IntersectionTest.addConnectionConnectionDuplicate (0.5633802816901409/0.5633802816901409)
IntersectionTest.addConnectionNegativeSpeedTest (0.5633802816901409/0.5633802816901409)
IntersectionTest.addConnectionTest (0.5633802816901409/0.5633802816901409)
IntersectionTest.addConnectionZeroSpeedTest (0.5633802816901409/0.5633802816901409)
IntersectionTest.getConnectedIntersectionsEmptyList (0.5633802816901409/0.5633802816901409)
IntersectionTest.getConnectedIntersectionsTest (0.5633802816901409/0.5633802816901409)
IntersectionTest.getConnectionDoesNotExistTest (0.5633802816901409/0.5633802816901409)
IntersectionTest.getConnectionExistsTest (0.5633802816901409/0.5633802816901409)

This will show you your manual style grade. To see detailed feedback, click on the question name in green (in this case, “Manual Style”) which will expand to show the grading rubric and deductions. Files with comments will have a speech box containing the number of comments in that file. To view the comments, click on the file name to expand it.

Submitted Files for Test Assignment

Results Code

- test/tms/intersection/IntersectionTest.java [Download](#)
- test/tms/sensors/DemoPressurePadTest.java [Download](#)
- src/tms/display/SimpleDisplay.java [Download](#)
- src/tms/intersection/Intersection.java [Download](#)
- src/tms/util/RouteNotFoundException.java [1 Comment](#) [Download](#)
- src/tms/util/TimedItemManager.java [Download](#)
- src/tms/util/TimedItem.java [Download](#)
- src/tms/util/DuplicateSensorException.java [Download](#)
- src/tms/route/TrafficLight.java [Download](#)
- src/tms/route/SpeedSign.java [Download](#)
- src/tms/route/Route.java [Download](#)

STUDENT
Ella Test

AUTOGRADER SCORE
72.5 / 85.0

QUESTION 2
Manual Style **11.0 / 15.0 pts**

Submitting to Gradescope

Using Gradescope you are able to submit files either by uploading them or through a GitHub repository.

To find the submission portal for Gradescope you will need to navigate to the dashboard for the subject where you will see a list of assignments.

The screenshot displays the Gradescope dashboard for CSSE2002/7023 Summer 2020. The interface includes a sidebar with navigation options like 'Dashboard' and 'Regrade Requests', and a list of assignments. The 'Test Assignment' is marked as 'Ungraded' and '20 Days, 5 Hours Late', with a release date of 'SEP 02'. The 'Test 2 Assignment' is marked as 'No Submission' and has a release date of 'NOV 09'. A progress bar for the 'Test 2 Assignment' indicates '49 minutes left' until the submission deadline.

NAME	STATUS	RELEASED	DUE (AEST)
Test Assignment	● Ungraded 20 Days, 5 Hours Late	Accepting late submissions SEP 02	Closes in 1 hour, 49 minutes OCT 20 AT 7:00AM LATE DUE DATE: NOV 09 AT 3:00PM
Test 2 Assignment	● No Submission	NOV 09	49 minutes left NOV 09 AT 2:00PM LATE DUE DATE: NOV 12 AT 2:00PM

By clicking on the assignment you wish to submit, the submission portal will appear.

Submit Programming Assignment

i Upload all files for your submission

SUBMISSION METHOD

Upload GitHub Bitbucket

DRAG & DROP

Any file(s) including .zip. Click to browse.

Upload

Cancel

File Upload

If you wish to upload a file, select the upload radio button and either drag the file from your computer or click on the drag & drop square to open a file explorer.

CSSE2002

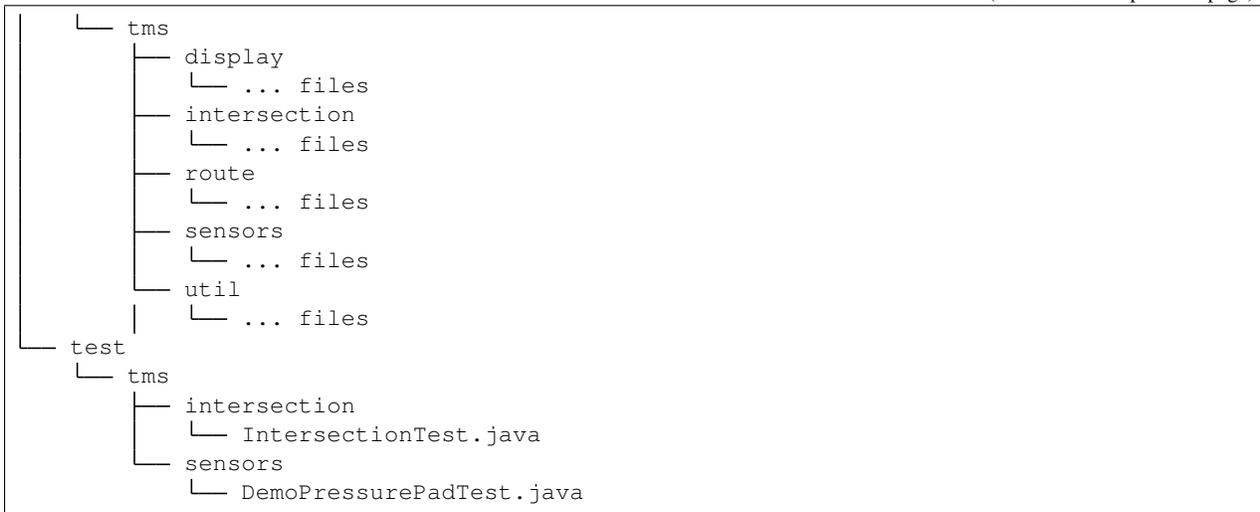
If you are submitting a file structure with directories you will need to zip them to preserve the structure.

Specifically for CSSE2002 you will need to zip your top level folders (src, test etc) together. In the below example you wouldn't compress a1, you need to select all of the sub folders of a1 (in this case src and test) and then compress them together.

```
a1
├── src
```

(continues on next page)

(continued from previous page)



GitHub

If you wish to submit an assignment that is in a GitHub repo you will need to select the GitHub radio button.

Submit Programming Assignment

i Upload all files for your submission

SUBMISSION METHOD

Upload
 GitHub
 Bitbucket

CONNECT YOUR ACCOUNT

You will then be prompted to connect your GitHub account to Gradescope.

Submit Programming Assignment

i Upload all files for your submission

SUBMISSION METHOD

Upload GitHub Bitbucket

REPOSITORY

Select a repository... ▼

BRANCH

Select a branch... ▼

Upload

Cancel

You will then be prompted to select the repository and branch of the assignment you wish to submit.

Submission

Once you have selected your options and clicked the Upload button, you will be taken to the results page. The results will not appear instantly as there may be some processing time. If your course has pre submission feedback you will be able to see it there.

1.4.2 Quickscope API

Details of the Quickscope API class, functions and methods.

quickscope

quickscope package

Subpackages

quickscope.server package

Submodules

quickscope.server.bundle module

`quickscope.server.bundle.copy_testrunner` (*bundle_directory*)

PythonEngine specific. Copies the testrunner.py testing utility package from ../templates into the included directory in the bundle.

Parameters `bundle_directory` (`Path`) – the temporary directory where the bundle is being created

Return type `None`

`quickscope.server.bundle.get_chalkbox` (*version*, *bundle_directory*)

Procures the specified version of ChalkBox from GitHub releases.

Parameters

- **version** (`str`) – the version of ChalkBox to get (e.g. v0.2.0)
- **bundle_directory** (`Path`) – the path to the temporary directory in which the bundle is constructed

Return type `Path`

Returns the path to the ChalkBox JAR in the temporary bundle directory.

`quickscope.server.bundle.get_dependencies` (*dependency_directory*)

Creates a list of dependencies based on the contents of the dependency directory.

Parameters `dependency_directory` (`Path`) – where the dependencies uploaded by the user are stored

Return type `List[str]`

Returns a list of the dependencies' paths (including the dependency directory)

`quickscope.server.bundle.produce_bundle` (*config*)

Performs bundle construction from the various elements based on the Engine and configuration specified.

Parameters `config` (`Dict[str, Any]`) – the configuration dictionary as prepared by `.templates.populate_config`

Return type `str`

Returns the path to the zipped bundle as a string

`quickscope.server.bundle.produce_config_file` (*form*, *bundle_directory*)

Takes the form from the React front-end and uses it, in combination with the engine default settings, to populate the configuration and write it to the `config.yaml` file in the bundle directory.

Parameters

- **form** (`Dict[str, Any]`) – the immutable form dictionary from the request object populated by the React front-end
- **bundle_directory** (`Path`) – the temporary directory where the bundle is being created

Return type `None`

`quickscope.server.bundle.produce_included_directory` (*source*, *bundle_directory*)

PythonEngine specific. Copies the entire 'included' directory with its uploaded components into the root of the bundle.

Parameters

- **source** (`Path`) – the original location of the included directory as uploaded by the user
- **bundle_directory** (`Path`) – the temporary directory where the bundle is being created

Return type `None`

`quickscope.server.bundle.produce_lib_directory` (*lib_directory*, *bundle_directory*)

JavaEngine specific. Gathers the lib file containing JAR dependencies and copies it to the temporary bundle directory.

Parameters

- **lib_directory** (`Path`) – the lib directory with JAR dependencies uploaded by the user
- **bundle_directory** (`Path`) – the temporary directory where the bundle is being created

Return type `None`

`quickscope.server.bundle.produce_resources_directory` (*resources_directory*, *bundle_directory*)

JavaEngine specific. Gathers the static resources directory and files uploaded by the user and copies them to the temporary bundle directory.

Parameters

- **resources_directory** (`Path`) – the static resources directory populated by the user's uploads
- **bundle_directory** (`Path`) – the temporary directory where the bundle is being created

Return type `None`

`quickscope.server.bundle.produce_run_script` (*run_call*, *bundle_directory=None*)

Creates the `run.sh` script in the root of the bundle that is used by Gradescope to start the autograding process.

Parameters

- **run_call** (`str`) – typically the call to start ChalkBox with whatever arguments are required
- **bundle_directory** (`Optional[Path]`) – the temporary directory where the bundle is being created

Return type `None`

`quickscope.server.bundle.produce_setup_script` (*setup_calls*, *bundle_directory*)

Creates the `setup.sh` script - required by Gradescope to prepare the Ubuntu environment - and places it in the bundle.

Parameters

- **setup_calls** (*str*) – the calls made to e.g. install packages, set the PATH etc. These come from `.templates.py`
- **bundle_directory** (*Path*) – the temporary directory where the bundle is being created

Return type *None*

`quickscope.server.bundle.produce_solution_directory` (*solution_directory*, *bundle_directory*)

Copies the solution directory uploaded by the user to the temporary bundle directory.

Parameters

- **solution_directory** (*Path*) – the directory containing the correct and faulty solutions uploaded by the user
- **bundle_directory** (*Path*) – the temporary directory where the bundle is being created

Return type *None*

`quickscope.server.bundle.reformat_test_classes` (*config*, *session_directory*)

JavaEngine specific. Transforms the assessable test classes listed in the config from Java import style (e.g. `chalkbox.import.style`) to path style (e.g. `chalkbox/import/style.java`).

Parameters

- **config** (*Dict[str, Any]*) – the config dictionary containing the test classes to update
- **session_directory** (*Path*) – the directory associated with the user's session where the user's uploads are stored

Return type *None*

`quickscope.server.config` module

`class quickscope.server.config.Config`

Bases: `object`

Configuration class for the quickscope Flask application.

`DEBUG = True`

`TESTING = True`

`UPLOAD_FOLDER = 'state'`

`quickscope.server.routes` module

`quickscope.server.routes.generate` ()

Generates the bundle based on the uploaded files and the configuration settings passed through in the form.

Return type *Response*

Returns a response that downloads the generated bundle to the client machine

`quickscope.server.routes.home` ()

Serve the React front end bundle.

Return type *str*

Returns the rendered template of the React `index.html` page

`quickscope.server.routes.upload_locations` (*component*)

Uploads a file to the correct location in the appropriate state directory (based on the session id and the component type).

Parameters `component` (`str`) – the component of the bundle that is being uploaded, this will determine the subdirectory in the state directory based on the engine

Return type `Response`

Returns a response object indicating success or failure

quickscope.server.run module

Utility class for running Quickscope

quickscope.server.templates module

`quickscope.server.templates.populate_config` (*config, form, session_directory, locations*)

Takes the user-provided configuration settings from the form and uses them to update the default values for the specified engine. User settings will override defaults.

Parameters

- **config** (`Dict[str, Any]`) – the basic settings common to all configurations: course code, assignment ID, engine, and session directory
- **form** (`Any`) – the form (immutable dictionary) attached to the Flask request containing the user-specified settings
- **session_directory** (`Path`) – the path to the directory associated with the user session matching the session ID created by the React front-end
- **locations** (`Dict[str, str]`) – the mapping from required components to their respective location in the bundle (e.g. `PYTHON_LOCATIONS` or `JAVA_LOCATIONS`, above)

Return type `Dict[str, Any]`

Returns the compiled configuration dictionary

quickscope.server.utils module

`quickscope.server.utils.collapse_path_overlap` (*clean_file, component, locations*)

Removes the risk of creating duplicated folders in the state directory by checking if the `clean_file` has any directory overlap with the component location e.g. `/solutions/correct/` and `correct/a1.py` would ensure that `a1.py` was put in `/solutions/correct/a1.py` and not `/solutions/correct/correct/a1.py`.

Parameters

- **clean_file** (`str`) – the uploaded file cleaned of any leading forward slashes
- **component** (`str`) – the particular component being uploaded
- **locations** (`Dict[str, str]`) – the set of upload locations specific to the engine associated with the session

Return type `str`

Returns the collapsed path as a string

`quickscope.server.utils.deep_update` (*original, updates*)

Update a nested dictionary with new values, leaving unupdated values in place. Modifies original in place.

Parameters

- **original** (Dict) – the dictionary whose values will be updated
- **updates** (Mapping) – the dictionary with the values to you want to insert into original

Return type Dict

`quickscope.server.utils.make_session` (*session_id*)

Creates a session directory with the given `session_id` if it does not already exist.

Parameters **session_id** (str) – the session ID generated by the React front-end whenever the page is loaded or reloaded.

Return type Path

Returns the path object corresponding to the `session_id`, which now must exist.

`quickscope.server.utils.reconstruct` (*session_id, component, files, locations*)

Finds the appropriate directory for uploaded file(s) and saves them there.

Parameters

- **session_id** (str) – the session ID provided by the React front end
- **component** (str) – the type of component being uploaded (e.g. linter config file)
- **files** – the files from the Flask request
- **locations** (Dict[str, str]) – the locations associated with the components that make up the engine selection

Return type None

Module contents

Submodules

`quickscope.build` module

`quickscope.build.build`()

Installs front-end dependencies and builds the front-end using yarn, which must already be installed.

Return type None

`quickscope.wsgi` module

Module contents

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

q

- `quickscope`, 36
- `quickscope.build`, 36
- `quickscope.server`, 36
 - `quickscope.server.bundle`, 32
 - `quickscope.server.config`, 34
 - `quickscope.server.routes`, 34
 - `quickscope.server.run`, 35
 - `quickscope.server.templates`, 35
 - `quickscope.server.utils`, 35
- `quickscope.wsgi`, 36

B

build() (in module *quickscope.build*), 36

C

collapse_path_overlap() (in module *quickscope.server.utils*), 35

Config (class in *quickscope.server.config*), 34

copy_testrunner() (in module *quickscope.server.bundle*), 32

D

DEBUG (*quickscope.server.config.Config* attribute), 34

deep_update() (in module *quickscope.server.utils*), 35

G

generate() (in module *quickscope.server.routes*), 34

get_chalkbox() (in module *quickscope.server.bundle*), 32

get_dependencies() (in module *quickscope.server.bundle*), 32

H

home() (in module *quickscope.server.routes*), 34

M

make_session() (in module *quickscope.server.utils*), 36

module

quickscope, 36

quickscope.build, 36

quickscope.server, 36

quickscope.server.bundle, 32

quickscope.server.config, 34

quickscope.server.routes, 34

quickscope.server.run, 35

quickscope.server.templates, 35

quickscope.server.utils, 35

quickscope.wsgi, 36

P

populate_config() (in module *quickscope.server.templates*), 35

produce_bundle() (in module *quickscope.server.bundle*), 32

produce_config_file() (in module *quickscope.server.bundle*), 32

produce_included_directory() (in module *quickscope.server.bundle*), 33

produce_lib_directory() (in module *quickscope.server.bundle*), 33

produce_resources_directory() (in module *quickscope.server.bundle*), 33

produce_run_script() (in module *quickscope.server.bundle*), 33

produce_setup_script() (in module *quickscope.server.bundle*), 33

produce_solution_directory() (in module *quickscope.server.bundle*), 34

Q

quickscope
module, 36

quickscope.build
module, 36

quickscope.server
module, 36

quickscope.server.bundle
module, 32

quickscope.server.config
module, 34

quickscope.server.routes
module, 34

quickscope.server.run
module, 35

quickscope.server.templates
module, 35

quickscope.server.utils
module, 35

quickscope.wsgi
module, 36

R

`reconstruct()` (in module `quickscope.server.utils`),
36

`reformat_test_classes()` (in module
`quickscope.server.bundle`), 34

T

`TESTING` (`quickscope.server.config.Config` attribute), 34

U

`UPLOAD_FOLDER` (`quickscope.server.config.Config` at-
tribute), 34

`upload_locations()` (in module
`quickscope.server.routes`), 34