

Automated Python Grading with GradeScope

Dr. Paul Urbik

December 1, 2023

Overview Slide

In this session we will learn how to set up a GradeScope for students to submit Python code for *automatic evaluation*.

In particular we cover,

1. Unit testing with `UnitTest`.
2. Configuring unit tests for GradeScope.
3. Manual marking with GradeScope.

Resources

The following are *web-links to resources* used throughout this talk.

1. UQTools tool.py.
2. unittest library.
3. plotcheker library.

§Setting the Assessment

Programming Assessments

Typically programming assessments will have two components:

1. A *specification sheet* specifying the assessment item *and* administrative items like due date, late policies, how to submit, etc.
2. A *framework* or *starter code*.

Specification Sheet

Question

Write a function

```
foo(x: int) -> int
```

which *doubles* its input.

Starter Code / Stub File / Framework

```
1  """
2  First Last      # update with your info
3  s00000000      # update with your info
4  """
5
6  def foo(x: int) -> int:
7      """ Return double the input.
8      >>> foo(1)
9      2
10     >>> foo(2)
11     4
12     """
13     pass
```

§Unit Testing

Unit Test

Unit testing is a type of software testing where *individual components* (e.g. functions and procedures) of a piece of software are tested.

A *unit test* is (perhaps) the simplest type of test one can perform.

UnitTest Library

The `unittest` library is a *unit testing framework* that is similar to other popular testing libraries (e.g. `Junit`).

It supports ...

1. test automation,
2. setup and shutdown code, and
3. aggregation of tests into collections.

test.py

```
1 import unittest
2 import submission
3
4 class TestFoo(unittest.TestCase):
5
6     def test_foo(self):
7         self.assertEqual(submission.foo(1), 2)
8
9 if __name__ == '__main__':
10     unittest.main()
```

This is student code. In this directory

Test group.

Single test.

Do this when calling file.

```
1 > ls
2 submission.py test.py
3 > python3 test.py
4 =====
5 FAIL: test_foo (__main__.TestFoo.test_foo)
6 -----
7 Traceback (most recent call last):
8   File "../test.py", line 7, in test_foo
9     self.assertEqual(submission.foo(1), 2)
10 AssertionError: None != 2
11
12 -----
13 Ran 1 test in 0.000s
14
15 FAILED (failures=1)
```

Student Submission

```
submission.py
1  """
2  Alice Liddle
3  s01234567
4  """
5
6  def foo(x: int) -> int:
7      """ Return double the input.
8
9      >>> foo(1)
10     2
11
12     >>> foo(2)
13     4
14
15     """
16     return 2*x
```

```
1 > ls
2 submission.py test.py
3 > python3 test.py
4 .
5 -----
6 Ran 1 test in 0.000s
7
8 OK
```

§ Moving Tests to GradeScope

Preparing GradeScope Files

In order to *streamline the process* of creating the various, and required, files for GradeScope Brae Webb wrote `tool.py` (UQTools) which automates the dirty work.

Supposing students were instructed to submit `submission.py` then...

1. Place `tool.py` in the same directory as your assignment files.
2. Create *at least one* file prefixed with `test_` comprised of unit tests that *imports all of* `tool.py`.
3. Do `>python3 tool.py` and follow prompts to obtain an `autograder.zip` file.


```
1 from tool import * This imports unittest
2 import submission
3
4 class TestFoo(TestCase):
5
6     def test_foo(self):
7         self.assertEqual(submission.foo(1), 2)
8
9 if __name__ == '__main__':
10     unittest.main()
```

```
1 > ls
2 submission.py test_foo.py  tool.py
3
4 > python3 test_foo.py
5 .
6 -----
7 Ran 1 test in 0.000s
8
9 OK
10
11 > python3 tool.py
12     test_foo (test_foo.TestFoo.test_foo) passed
13 Generating autograder.zip
14     Found extra file .DS_Store in directory, should this be included? (y/n) n
15     Not including .DS_Store
16     Found extra file __pycache__ in directory, should this be included? (y/n) n
17     Not including __pycache__
18
19 > ls
20 autograder.zip submission.py  test_foo.py  tool.py
```

Preparing GradeScope (Create Assignment)

The screenshot shows a web browser at gradescope.com. The left sidebar contains the GradeScope logo and a 'Your Courses' section with a welcome message: 'Welcome to Gradescope! Click on one of your courses to the right, or on the Account menu below.' The main content area is titled 'Your Courses' and shows a list for 'Summer 2023'. There are three course cards: 1) 'CSSE1001S_7360_61263' with details '[CSSE1001/7030] Introduction to Software Engineering (St Lucia), Semester 2, 2023' and a dark teal bar indicating '5 assignments'. 2) 'DEMO1001' with details 'Demo Course' and 'No Published Grades', and a light blue bar indicating '1 assignment'. 3) A grey card with a plus sign and the text '+ Create a new course'.

gradescope.com

gradescope
by Turnitin

Your Courses

Welcome to Gradescope!
Click on one of your courses
to the right, or on the
Account menu below.

Your Courses

Summer 2023

CSSE1001S_7360_61263
[CSSE1001/7030]
Introduction to Software
Engineering (St Lucia).
Semester 2, 2023

5 assignments

DEMO1001
Demo Course
No Published Grades

1 assignment

+ Create a new course

Preparing GradeScope (Create Assignment)

DEMO1001

Summer 2023

Entry Code: ZWBEKZ

Course ID: 674501

Description

Demonstration purposes.

Things To Do

- 1 Create your first assignment from the [Assignments](#) page.

↕ Active Assignments

Released

Due (AEST) ▼

↕ Submissions

% Graded ↕

Published

Regrades

You currently have no assignments.

Create an assignment to get started.

Create Assignment

Preparing GradeScope (Programming Assignment)

Create Assignment

1 Assignment Type

2 Assignment Settings

← Exit

Assignment Types

📄 Exam / Quiz

📄 Homework / Problem Set

📄 Bubble Sheet

📄 **Programming Assignment**

📄 Online Assignment **Beta**



Select an Assignment Type

Gradescope supports a variety of paper-based, online, and code assignments. Click on one to learn more.

Create Assignment

Assignment Type

2 Assignment Settings

[Go Back](#)

Assignment Type

[Programming Assignment](#)

* Required fields

Assignment Name *

Your First Programming Assessment

Submission Anonymization

Enable anonymous grading

Hide identifiable student information from being listed with submissions.

Autograder Points *

10

Manual Grading

Enable manual grading

Release Date * (AEST)

2023-11-27, 02:44 PM

Due Date * (AEST)

2023-12-02, 02:44 PM

Allow late submissions

Late Due Date (AEST)

yyyy-mm-dd, --:-- --

Group Submission

Enable group submission

Limit Group Size:

No Max

Create your Rubric

Before student submission

While grading submissions

Leaderboard

Enable leaderboard

Default # Of Entries

No Max



Create Assignment

Manual Grading Rubric (Optional)

Outline for Your First Programming Assessment

12 points total

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

#	Title	Points
1	Autograder	10.0
2	PEP8	2
2.1	Variables names	1
2.2	Line Endings	1

[+ New Question](#)

Cancel

Save Outline

Upload Autograder



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.

* Required field

Autograder Configuration

Zip file upload Manual Docker configuration

Autograder *

Please select a file

Select Autograder (.zip)



tool.py creates this zip for you

Base Image OS

Ubuntu

Base Image Version

22.04

Base Image Variant

Base

Choose the [base image](#) that will be used to build your autograder. This determines the operating system version and packages available in your autograder.

Update Autograder



hit the button

Docker Image Status

built as of Nov 30, 2023 at 3:04:54 PM AEST

Build Output

```
Get:7 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [44.0 kB]
Get:8 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1265 kB]
Get:9 http://archive.ubuntu.com/ubuntu jammy/restricted amd64 Packages [164 kB]
Get:10 http://archive.ubuntu.com/ubuntu jammy/main amd64 Packages [1792 kB]
Get:11 http://archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [1520 kB]
Get:12 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1292 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 Packages [49.8 kB]
Get:14 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1535 kB]
Get:15 http://archive.ubuntu.com/ubuntu jammy-backports/universe amd64 Packages [32.6 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy-backports/main amd64 Packages [78.3 kB]
Get:17 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [1027 kB]
Get:18 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [1494 kB]
```

Fetched 28.6 MB in 2s (11.5 MB/s)

Reading package lists...

```
+ bash /autograder/source/setup.sh
```

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Reading package lists...

Building dependency tree...

Reading state information...

python3 is already the newest version (3.10.6-1-22.04).

0 upgraded, 0 newly installed, 0 to remove and 30 not upgraded.

```
+ apt-get clean
```

```
+ rm -rf /var/lib/apt/lists/archive.ubuntu.com_ubuntu_dists_jammy-backports_InRelease /var/lib/apt/lists/
```

Removing intermediate container 63d96260ecle

```
---> 019f1dfefaa
```

Successfully built 019f1dfefaa

Successfully tagged gradescope/autograders:us-prod-docker_image-313543

Build Errors

once successfully built then...

Test the Autograder (Instructor View)

Base Image OS: Ubuntu

Base Image Version

Base Image Variant

Submit Programming Assignment

Upload all files for your submission

Submission Method

Upload GitHub Bitbucket

Add files via Drag & Drop or [Browse Files](#).

Name	Size	Progress	
submission.py	0.2 KB	<div style="width: 10%;"></div>	✖

Cancel Upload

Get:7 <http://security.ubuntu.com/ubuntu> jammy-security/universe amd64 Packages [1027 kB]
Get:8 <http://security.ubuntu.com/ubuntu> jammy-security/restricted amd64 Packages [1494 kB]
Get:9 <http://archive.ubuntu.com/ubuntu> jammy-updates/multiverse amd64 Packages [49.8 kB]
Get:10 <http://archive.ubuntu.com/ubuntu> jammy-updates/main amd64 Packages [1535 kB]
Get:11 <http://archive.ubuntu.com/ubuntu> jammy-backports/universe amd64 Packages [32.6 kB]
Get:12 <http://archive.ubuntu.com/ubuntu> jammy-backports/main amd64 Packages [78.3 kB]
Get:13 <http://archive.ubuntu.com/ubuntu> jammy-updates/multiverse amd64 Packages [49.8 kB]
Get:14 <http://archive.ubuntu.com/ubuntu> jammy-updates/main amd64 Packages [1535 kB]
Get:15 <http://archive.ubuntu.com/ubuntu> jammy-backports/universe amd64 Packages [32.6 kB]
Get:16 <http://archive.ubuntu.com/ubuntu> jammy-backports/main amd64 Packages [78.3 kB]
Get:17 <http://security.ubuntu.com/ubuntu> jammy-security/universe amd64 Packages [1027 kB]
Get:18 <http://security.ubuntu.com/ubuntu> jammy-security/restricted amd64 Packages [1494 kB]
Fetched 28.6 MB in 2s (11.5 MB/s)
Reading package lists...

Test the Autograder (Instructor View)

Autograder Results

Results

Code

Autograder Output (hidden from students)

.

Ran 1 test in 0.000s

OK

test_foo (test_foo.TestFoo) (1/1)

Your First Programming Assessment

● Graded

Student

Unknown Student (removed from roster?)

Total Points

1 / 12 pts

Autograder Score

1.0 / 10.0

Passed Tests

test_foo (test_foo.TestFoo) (1/1)

Bells and Whistles

We have created the *absolute most basic* autograder (one function and one test).

Let us now learn how to make a more robust tester with *multiple tests* of *different weights* with different *visibility* (e.g. see result immediately versus after grade release).

```
1 class TestFoo(unittest.TestCase):
2     @weight(10)                Number of points for this test.
3     @number("1")              Can be given in any order.
4     @visibility('after_published')
5         When students should see test outcome, omit for immediate.
6     def test_foo(self):
7         """ Student sees this.
8         """
9         self.assertEqual(submission.foo(1), 2)
```

Test the Autograder (Instructor View)

Autograder Results

Results

Code

Autograder Output (hidden from students)

```
.  
-----  
Ran 1 test in 0.000s  
  
OK
```

1) Student sees this. (10/10)

^ Test Number



Hidden Test

Your First Programming Assessment

● Graded

Student

Unknown Student (removed from roster?)

Total Points

10 / 12 pts

Autograder Score

10.0 / 10.0

Passed Tests

1) Student sees this. (10/10)

Let us add another test to TestFoo...

```
1 class TestFoo(unittest.TestCase):
2     .
3     .
4     .
5     @weight(5)
6     @number("2")
7     def test_foo_again(self):
8         """ foo(5) should be 10
9         """
10        self.assertEqual(submission.foo(5), 10)
```

Can be given in any order.

Instructor View – Test Autograder

Autograder Results

Results

Code

Autograder Output (hidden from students)

..

Ran 2 tests in 0.000s

OK



1) foo(1) should be 2 (10/10) 

2) foo(5) should be 10 (5/5)

Your First Programming Assessment

● Graded

Student

Unknown Student (removed from roster?)

Total Points

15 / 12 pts

Autograder Score

15.0 / 10.0

Passed Tests

1) foo(1) should be 2 (10/10)

2) foo(5) should be 10 (5/5)

Student View (After Submission and Before Deadline)

Autograder Results

Results

Code

2) foo(5) should be 10 (5/5)

Your First Programming Assessment

● Ungraded

Student

Alice Liddle

Total Points

- / 12 pts

Autograder Score

- / 10.0

Passed Tests

2) foo(5) should be 10 (5/5)

§AutoGrading IO Sessions

Question

Write a function

`foo()` \rightarrow `None`:

that

1. Prompt the user until they enter a positive integer.
2. Prints the integer (say n) n -many times.
3. Quits if `q` is entered and prints **Goodbye!** but otherwise repeats.

```
1 def foo() -> None:
2     """ Implements foo according to the spec-sheet.
3     >>> foo()
4     Prompt: X
5     Prompt: 3
6     333
7     Prompt: 0
8     Prompt: 1
9     1
10    Prompt: q
11    Goodbye!
12
13    """
14    pass
```

```
1 from tool import *
2 class TestFoo(unittest.TestCase):
3     @weight(1)
4     def test_foo(self):
5         """ The doc-string example from spec.
6         """
7
8         def test_foo(self):
9             inp = ['X', '3', '0', '1', 'q']           Stream of input
10            io = 'io.txt'                             Entire IO session
11            self.assertIOFromFileEquals(submission.foo,
12                                       'inp.txt',
13                                       'io.txt')
```

1 Prompt: X

2 Prompt: 3

3 333

4 Prompt: 0

5 Prompt: 1

6 1

7 Prompt: q

8 Goodbye!

9

§AutoGrading Matplotlib

```
1 fig, axes = plt.subplots(1, 3)
2 x, y = np.random.rand(20), np.random.rand(20)
3
4 create a scatter plot with plot
5 axes[0].plot(x, y, 'o', color='b', ms=5)
6
7 create a scatter plot with scatter
8 axes[1].scatter(x, y, s=25, c='b')
9
10 create a scatter plot with plot and scatter!
11 axes[2].plot(x[:10], y[:10], 'o', color='b', ms=5)
12 axes[2].scatter(x[10:], y[10:], s=25, c='b')
13
14 fig.set_size_inches(12, 4)
```

```
1 from plotchecker import ScatterPlotChecker
```

```
2
```

```
3 for ax in axes: run the same assertions on all the plots!
```

```
4     pc = ScatterPlotChecker(ax)
```

```
5     pc.assert_x_data_equal(x)
```

```
6     pc.assert_y_data_equal(y)
```

```
7     pc.assert_colors_equal('b')
```

```
8     pc.assert_edgecolors_equal('b')
```

```
9     pc.assert_edgewidths_equal(1)
```

```
10    pc.assert_sizes_equal(25)
```

```
11    pc.assert_markersizes_equal(5)
```

```
12    pc.assert_alphas_equal(1.0)
```

```
13
```

```
14 print('Success!')
```

Questions?