Automated Python Grading with GradeScope

Dr. Paul Vrbik

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 Assesments

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

submission.py

```
"""
First Last    # update with your info
s00000000     # update with your info
"""

def foo(x: int) -> int:
    """ Return double the input.
    >>> foo(1)
    2
    >>> foo(2)
    4
    """
    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 perfom.

`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                    This is student code. In this directory
3
4  class TestFoo(unittest.TestCase):                              Test group.
5
6      def test_foo(self):                                        Single test.
7          self.assertEqual(submission.foo(1), 2)
8
9  if __name__ == '__main__':                          Do this when calling file.
10     unittest.main()
```

```
> ls
submission.py test.py
> python3 test.py
======================================================================
FAIL: test_foo (__main__.TestFoo.test_foo)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "../test.py", line 7, in test_foo
    self.assertEqual(submission.foo(1), 2)
AssertionError: None != 2

----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (failures=1)
```

Student Submission

submission.py

```
"""
Alice Liddle
s01234567
"""

def foo(x: int) -> int:
    """ Return double the input.
    >>> foo(1)
    2
    >>> foo(2)
    4
    """
    return 2*x
```

```
> ls
submission.py test.py
> python3 test.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

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...

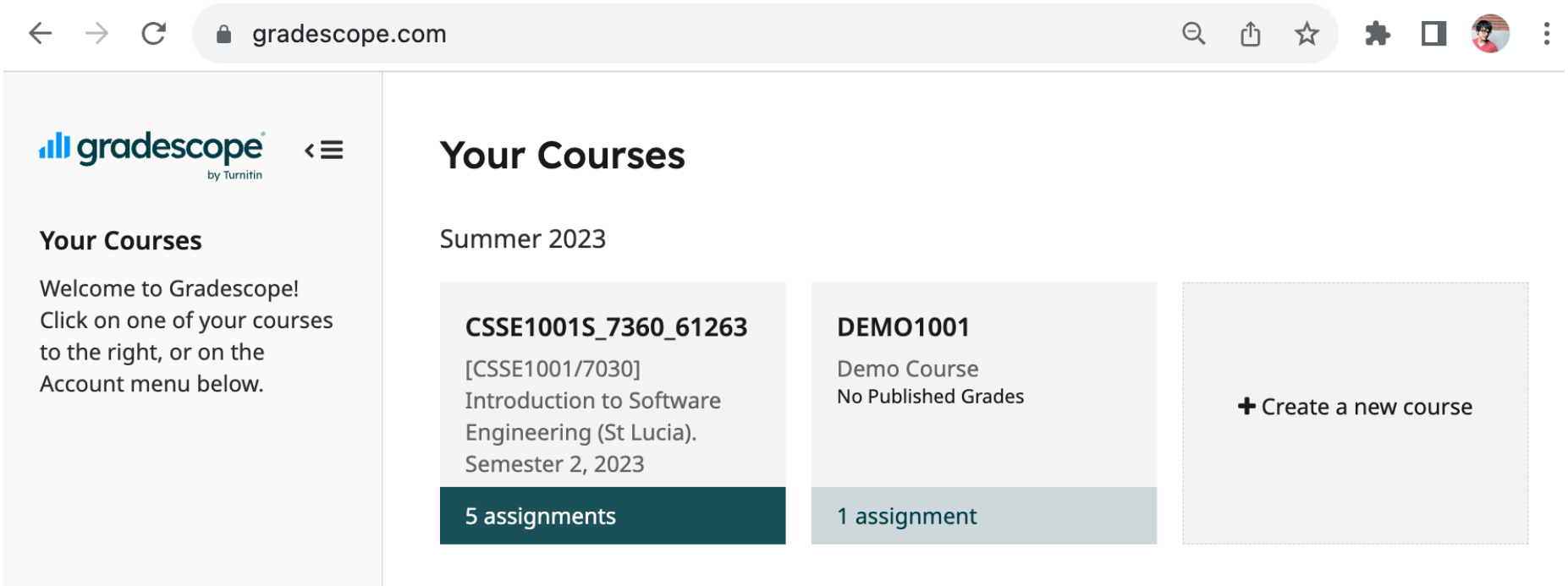
1. Place `tool.py` in the same directory as your assignment files.
2. Create *at least one* file prefixed with `test_` comprised of unit tests that *imports all of* `tool.py`.
3. Do `>python3 tool.py` and follow prompts to obtain an `autograder.zip` file.

test.py

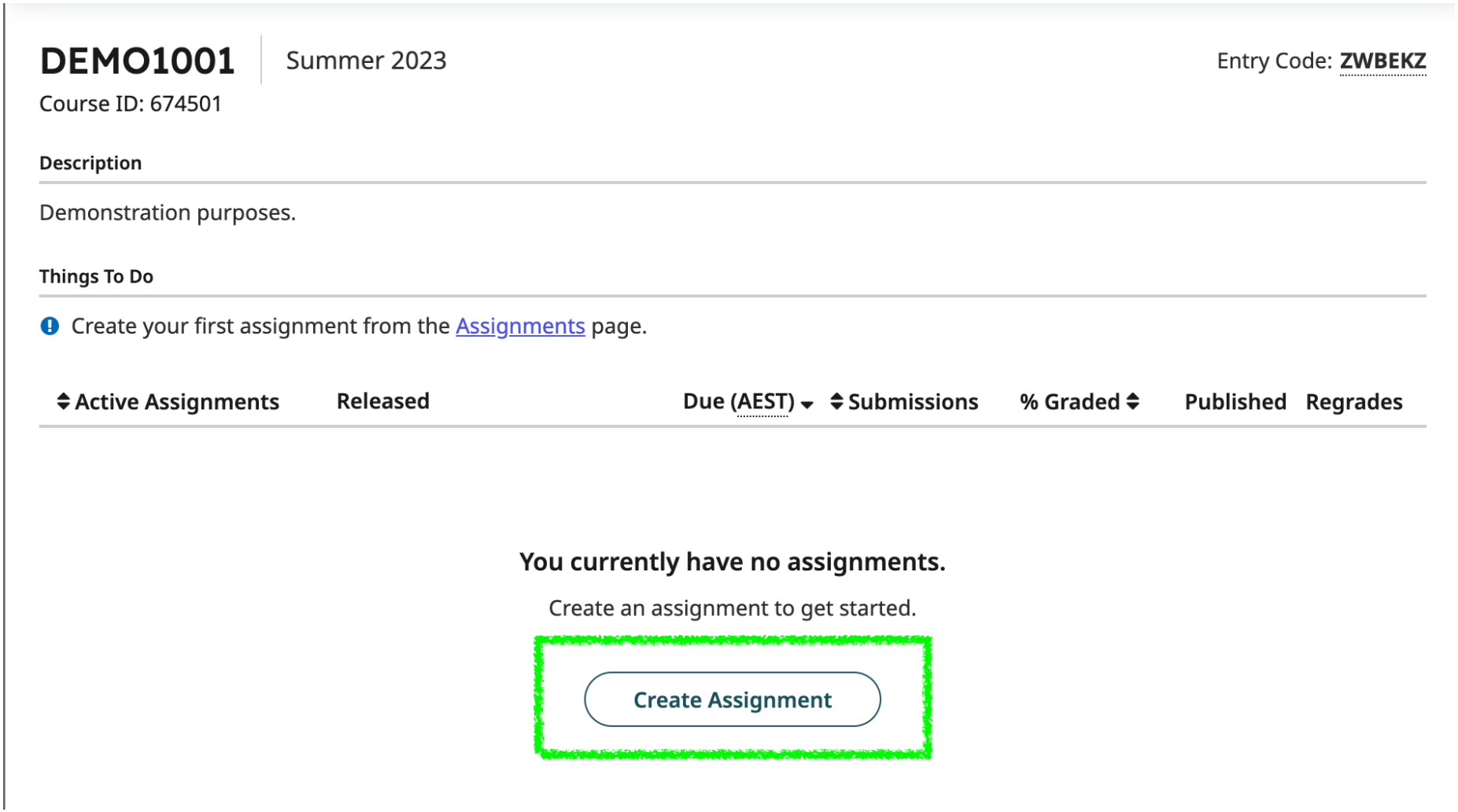
```
1  from tool import *                                  This imports unittest
2  import submission
3
4  class TestFoo(TestCase):
5
6      def test_foo(self):
7          self.assertEqual(submission.foo(1), 2)
8
9  if __name__ == '__main__':
10     unittest.main()
```

```
> ls
submission.py test_foo.py   tool.py

> python3 test_foo.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK

> python3 tool.py
        test_foo (test_foo.TestFoo.test_foo) passed
Generating autograder.zip
        Found extra file .DS_Store in directory, should this be included? (y/n) n
        Not including .DS_Store
        Found extra file __pycache__ in directory, should this be included? (y/n) n
        Not including __pycache__

> ls
autograder.zip submission.py  test_foo.py    tool.py
```

Preparing GradeScope (Create Assignment)



Preparing GradeScope (Create Assignment)



Preparing GradeScope (Programming Assignment)

Create Assignment

1 Assignment Type — 2 Assignment Settings

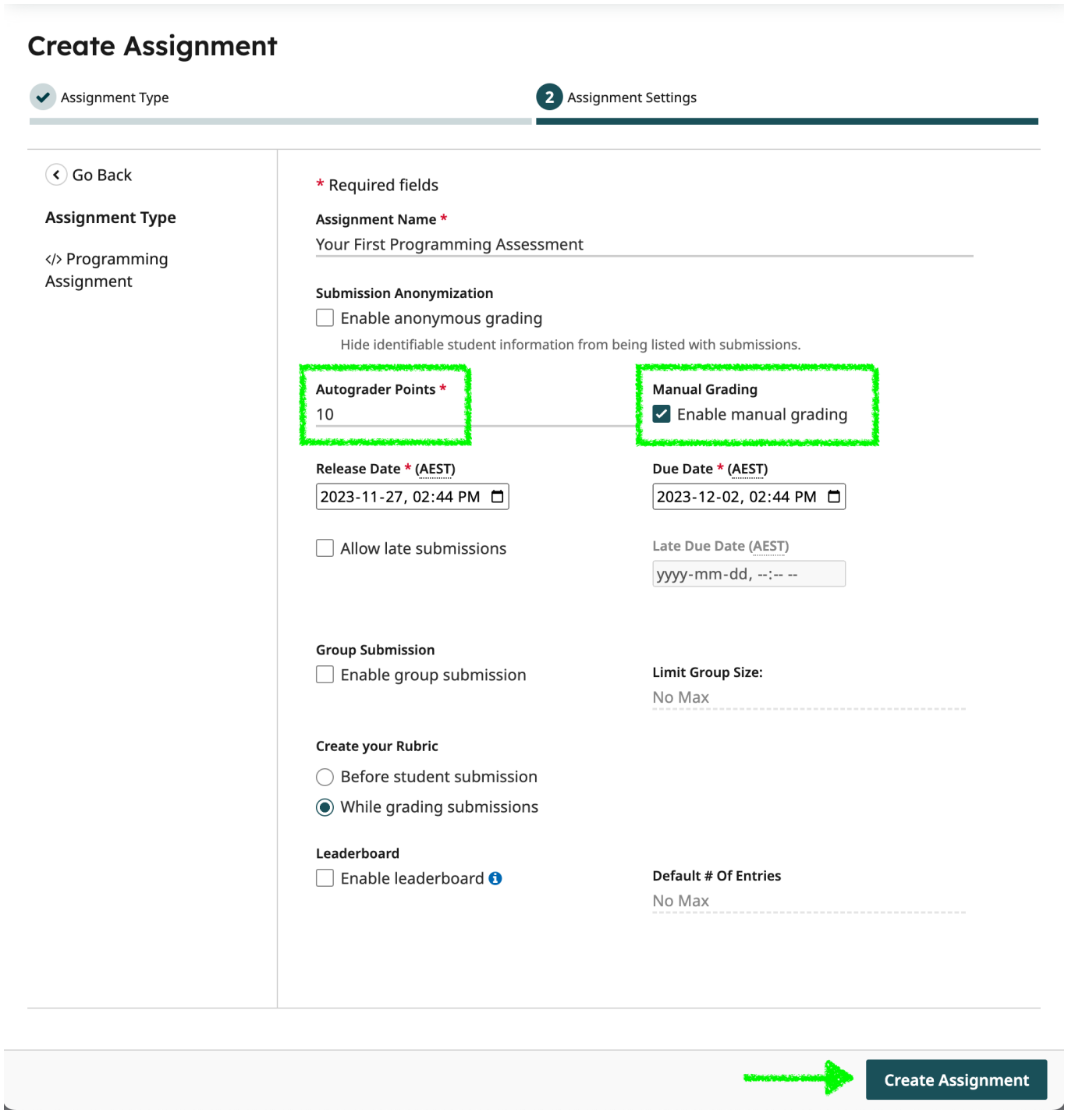
Exit

Assignment Types

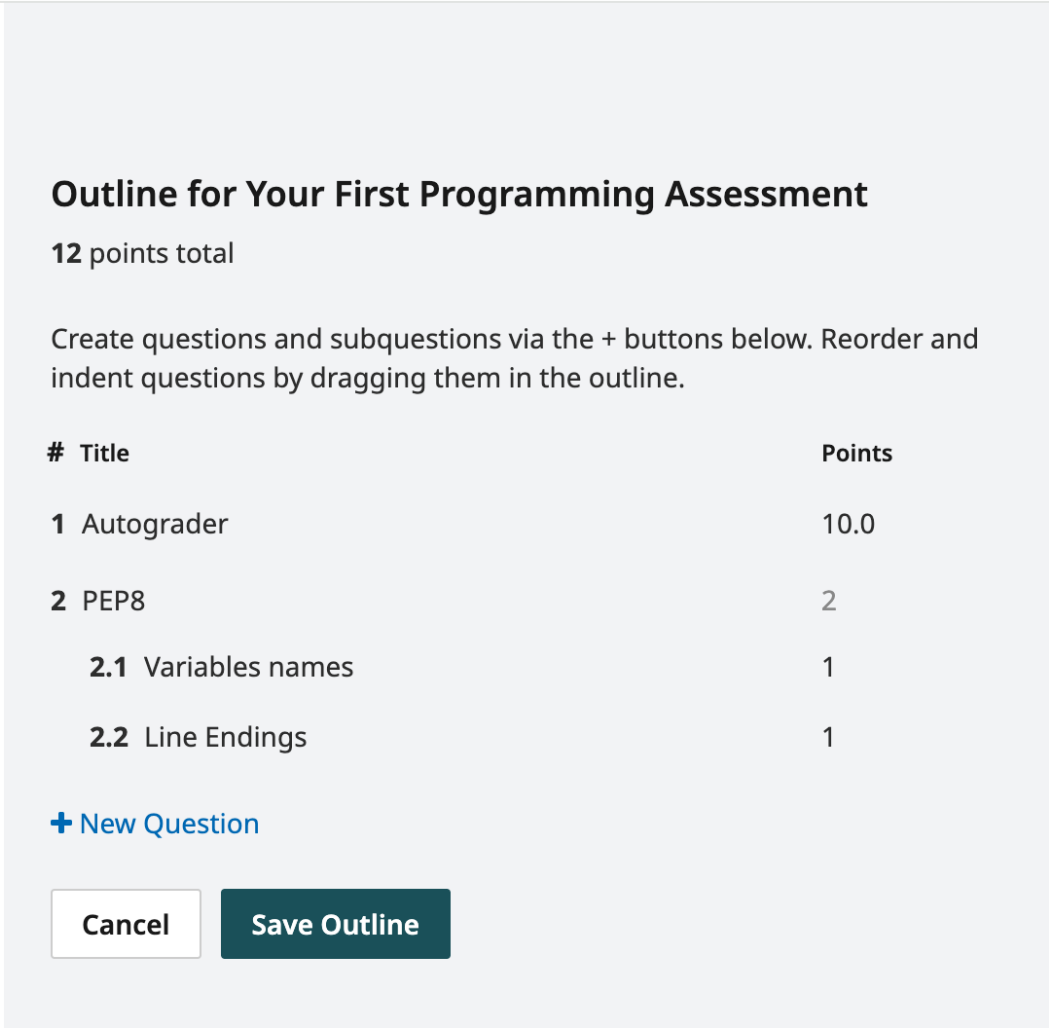
- Exam / Quiz
- Homework / Problem Set
- Bubble Sheet
- </> Programming Assignment
- Online Assignment **Beta**

Select an Assignment Type

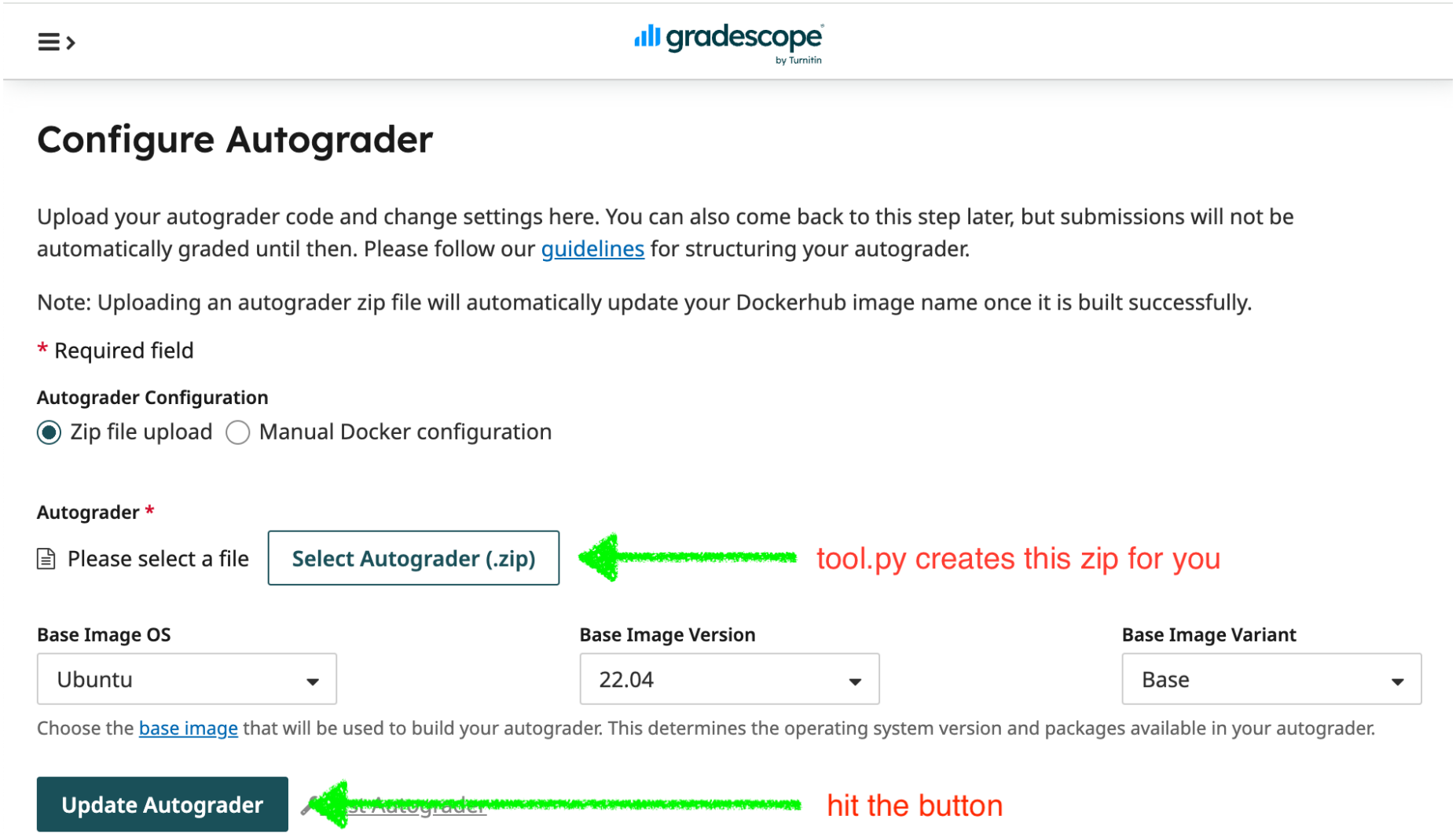
Gradescope supports a variety of paper-based, online, and code assignments. Click on one to learn more.


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)



Upload Autograder



Update Autograder | Test Autograder

Docker Image Status

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

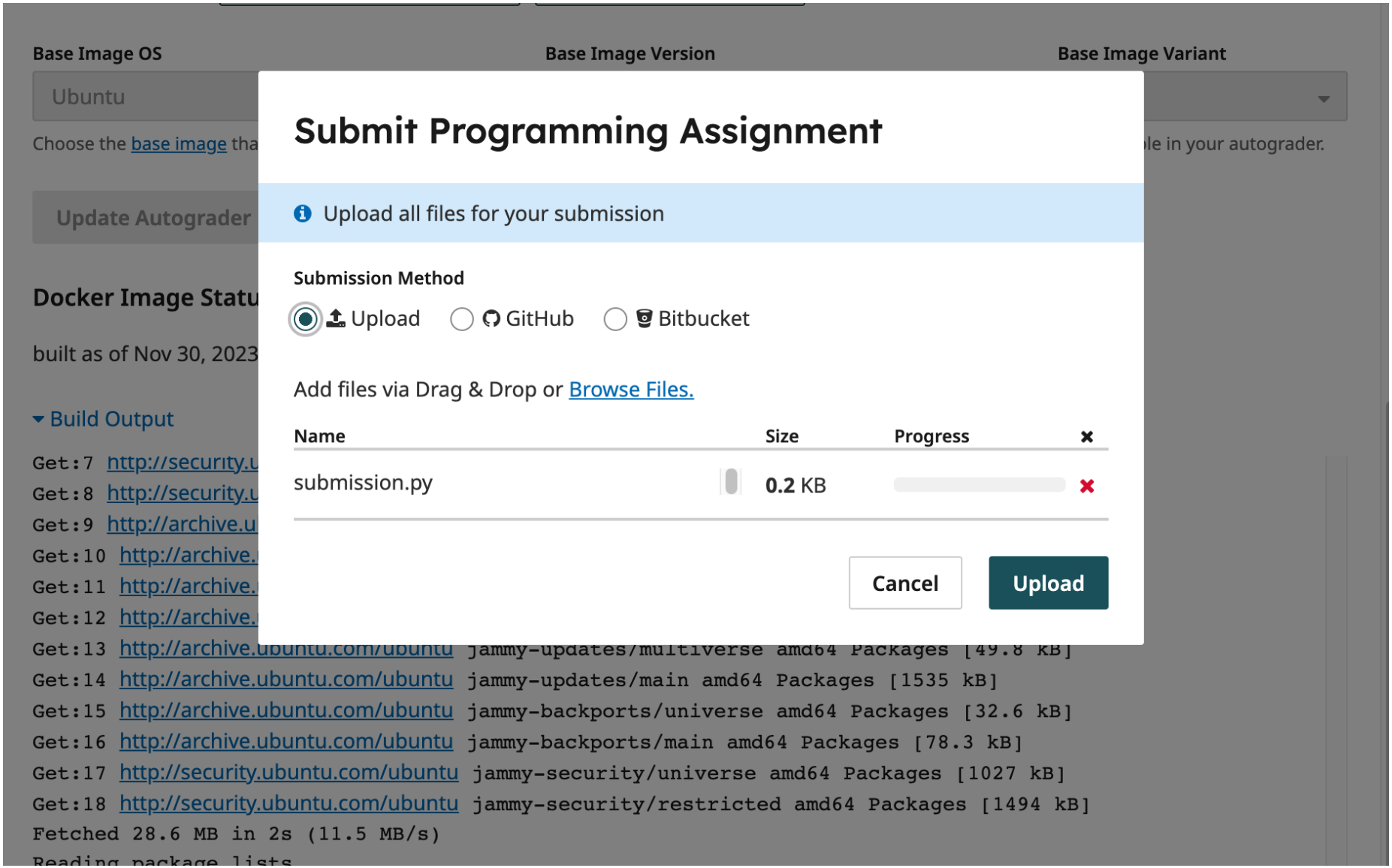
▾ Build Output

```
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 63d96260ec1e
 ---> 019f1dfefaab
Successfully built 019f1dfefaab
Successfully tagged gradescope/autograders:us-prod-docker_image-313543
```

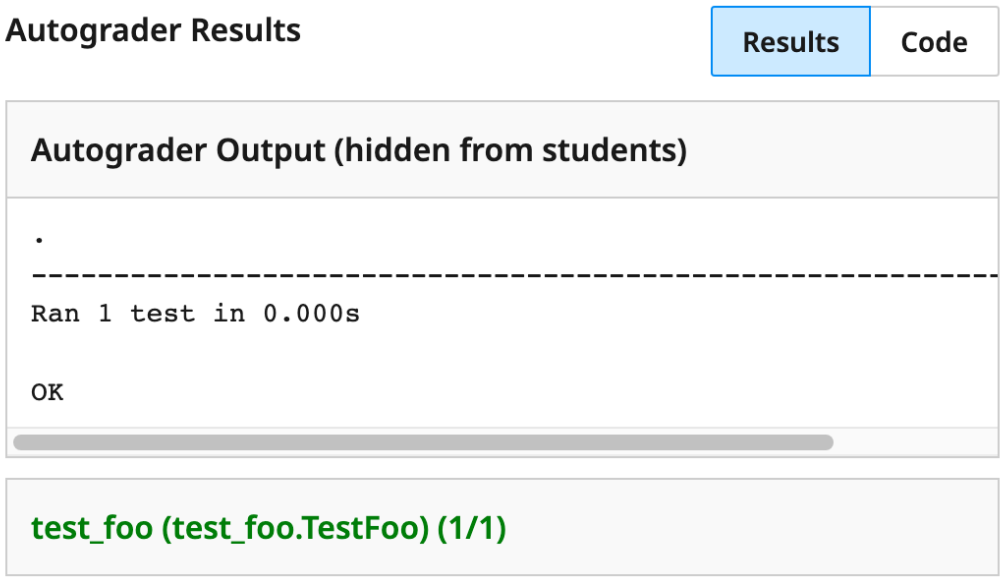
once successfully built then...

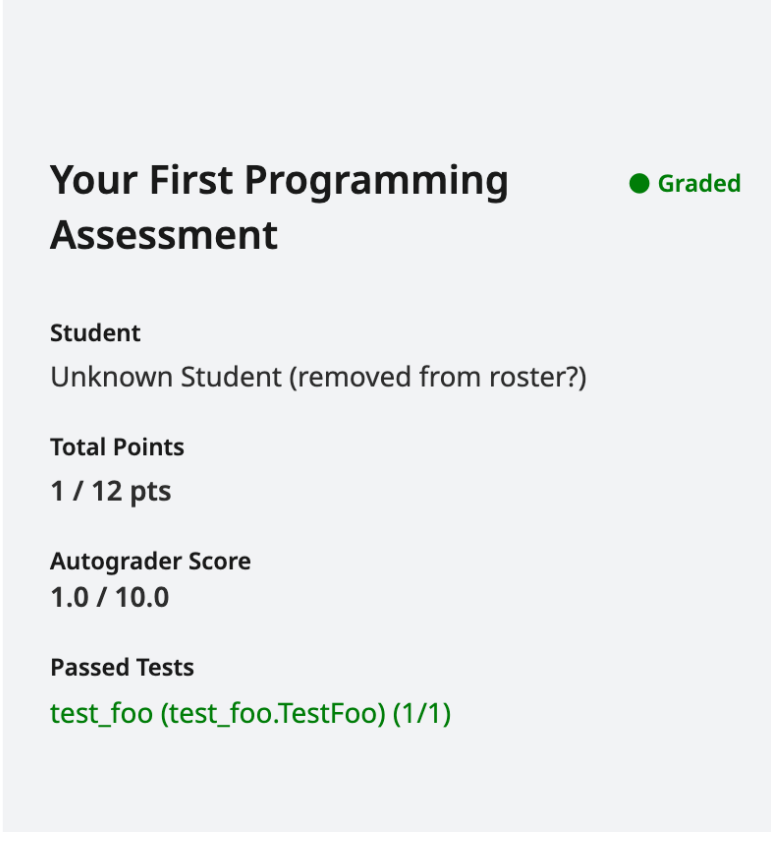
▾ Build Errors

Test the Autograder (Instructor View)



Test the Autograder (Instructor View)





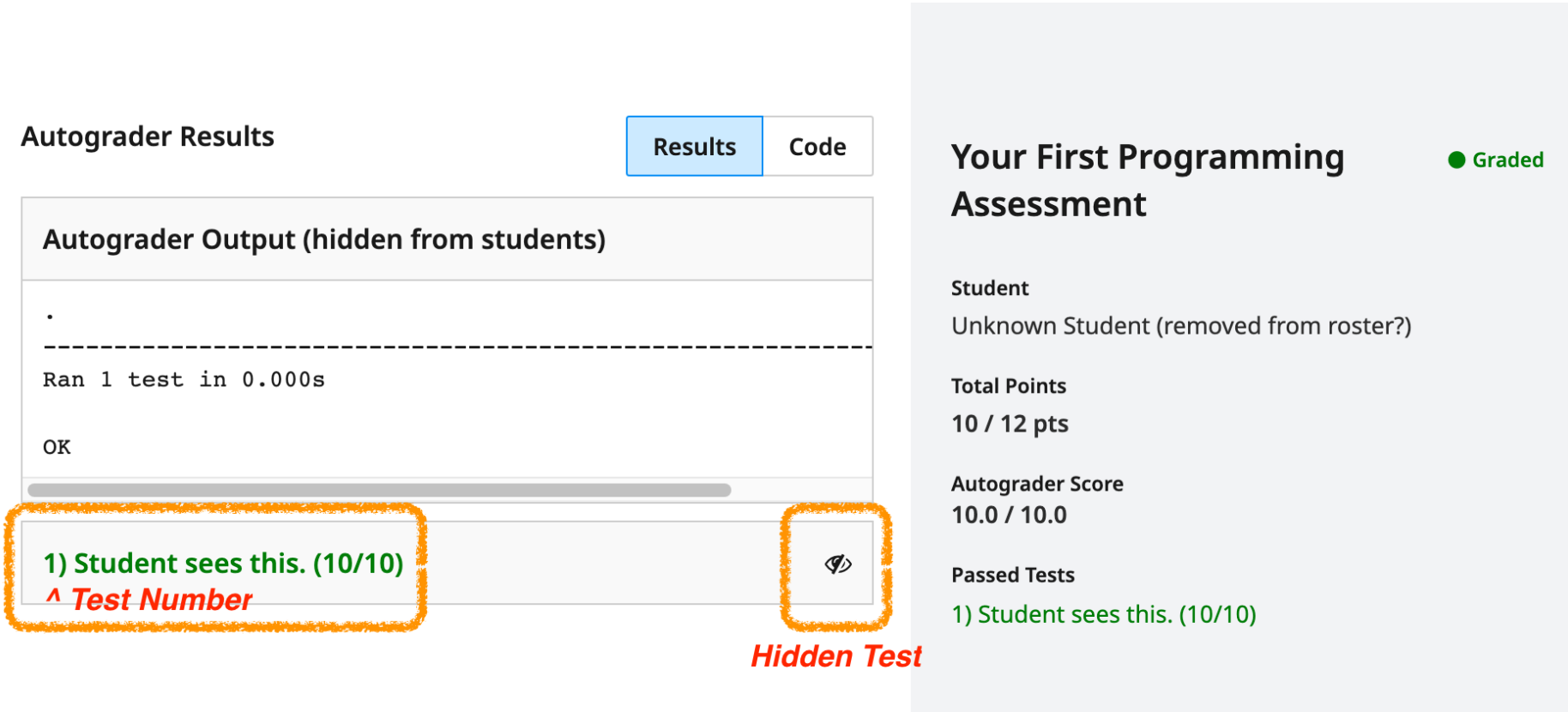
Bells and Whistles

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)
```

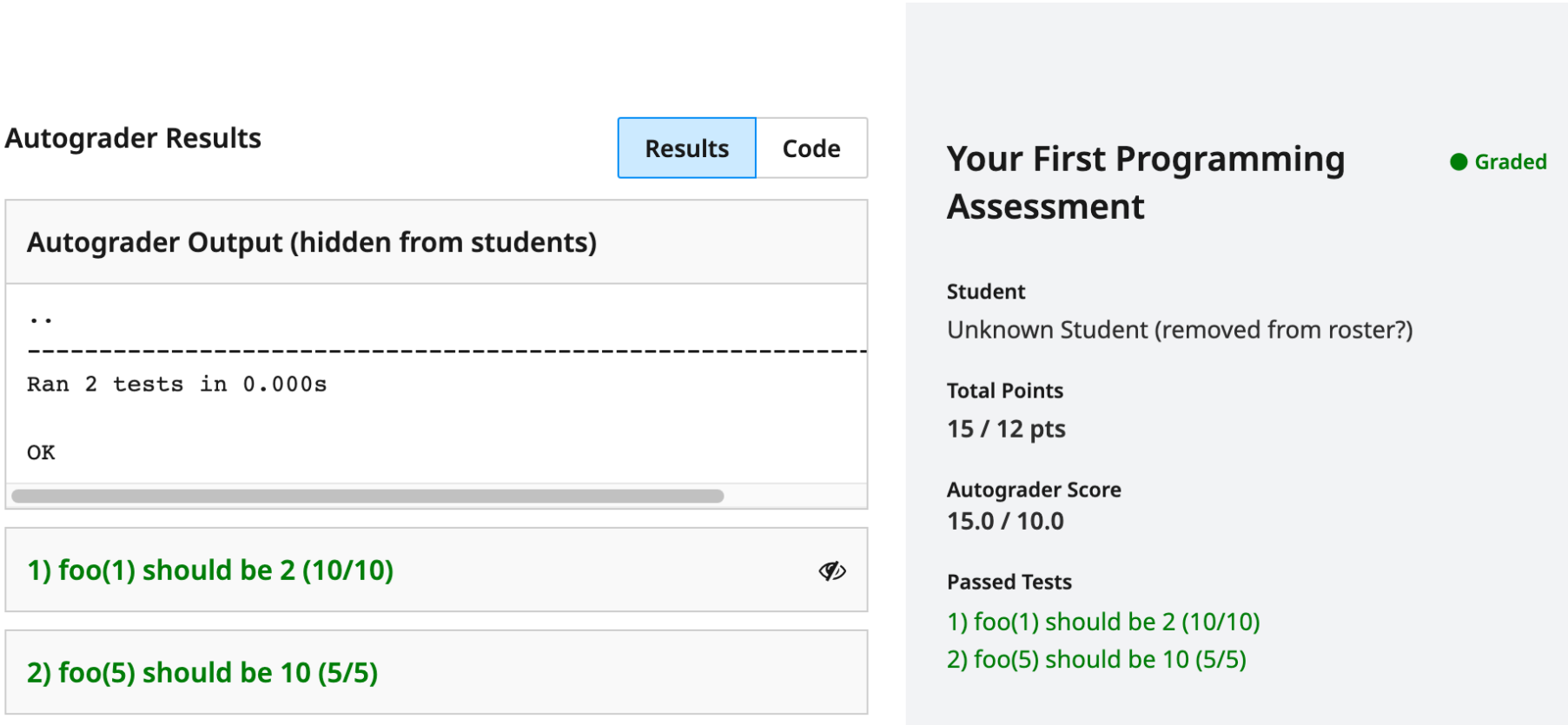
Test the Autograder (Instructor View)



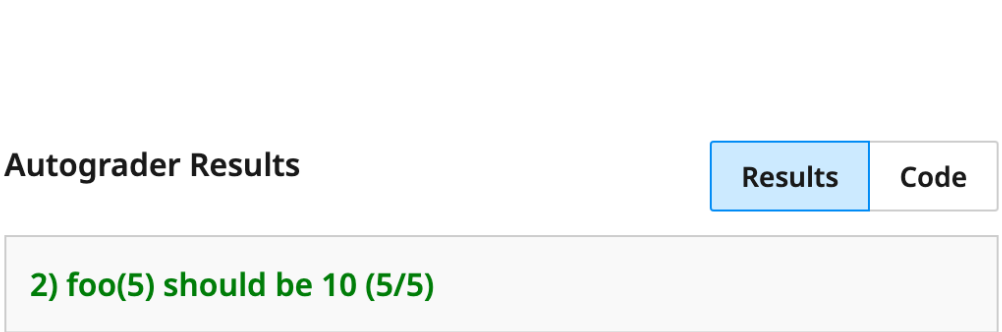
Let us add another test to `TestFoo`...

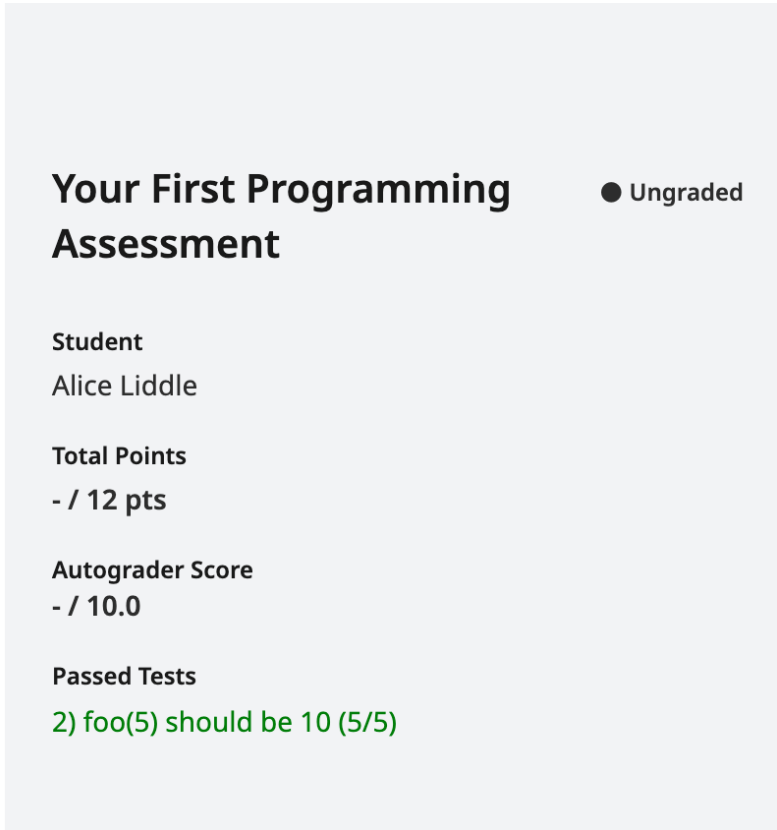
```
1  class TestFoo(unittest.TestCase):
2      .
3      .
4      .
5      @weight(5)
6      @number("2")                    Can be given in any order.
7      def test_foo_again(self):
8          """ foo(5) should be 10
9          """
10         self.assertEqual(submission.foo(5), 10)
```

Instructor View – Test Autograder



Student View (After Submission and Before Deadline)





§AutoGrading IO Sessions

Question

Write a function

```
foo() -> 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.

```
def foo() -> None:
    """ Implements foo according to the spec-sheet.
    >>> foo()
    Prompt: X
    Prompt: 3
    333
    Prompt: 0
    Prompt: 1
    1
    Prompt: q
    Goodbye!

    """
    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')
```

io.txt

```
Prompt: X
Prompt: 3
333
Prompt: 0
Prompt: 1
1
Prompt: q
Goodbye!

```

§AutoGrading MatPlotLib

```
fig, axes = plt.subplots(1, 3)
x, y = np.random.rand(20), np.random.rand(20)

create a scatter plot with plot
axes[0].plot(x, y, 'o', color='b', ms=5)

create a scatter plot with scatter
axes[1].scatter(x, y, s=25, c='b')

create a scatter plot with plot and scatter!
axes[2].plot(x[:10], y[:10], 'o', color='b', ms=5)
axes[2].scatter(x[10:], y[10:], s=25, c='b')

fig.set_size_inches(12, 4)
```

```
from plotchecker import ScatterPlotChecker

for ax in axes:                    run the same assertions on all the plots!
    pc = ScatterPlotChecker(ax)
    pc.assert_x_data_equal(x)
    pc.assert_y_data_equal(y)
    pc.assert_colors_equal('b')
    pc.assert_edgecolors_equal('b')
    pc.assert_edgewidths_equal(1)
    pc.assert_sizes_equal(25)
    pc.assert_markersizes_equal(5)
    pc.assert_alphas_equal(1.0)

print('Success!')
```

Questions?