UNIVERSITY OF MIAMI
DEPARTMENT of INFORMATION TECHNOLOGY

Programming Assignment with Gradescope Autograder

Creating a Programming Assignment with Autograder

This documentation outlines setting up an autograder for unit and diff-style testing, along with several test cases. Additional autograders are available on the Gradescope GitHub for your reference: https://github.com/gradescope/autograder_samples

The Python autograder example showcases a variety of test cases. For instance, the 'TestFiles' class checks for submitted files, ensuring all required files are present. Additionally, the 'TestIntegration' class conducts integration tests on the script, evaluating single input expressions and checking the functionality to quit the program.
https://github.com/gradescope/autograder_samples/tree/master/python

Example Assignments with Autograder

The autograder supports Differential and Unit testing. **Unit testing** focuses on individual functions or methods, ensuring a focused evaluation of critical components. **For differential testing**, the entire program runs in a Python subprocess, enabling comprehensive input-output testing. This approach allows for a broader assessment, as it examines the program's behavior as a whole.

Example Assignment Description for Unit Testing: You'll be tasked with creating a class called **Fibonacci**. This class should have a method named **Fibonacci(n)** that generates the Fibonacci sequence up to n elements. Additionally, you'll need to handle potential exceptions using a custom exception class called **FibonacciException**. Submit your code in a file named '**fib.py**'.

```
import re

class FibonacciException(Exception):
    """A class to throw if there are issues with the input."""
    def __init__(self, value):
        self.value = value

    def __str__(self):
        return repr(self.value)


class Fibonacci(object):
    """Generate Fibonacci sequence up to n elements."""

    def fibonacci(self, n):
        *IMPLEMENT*

if __name__ == "__main__":
    fib = Fibonacci()
    try:
        n = int(input("Enter the length of the Fibonacci sequence: "))
        print(fib.fibonacci(n))
    except ValueError:
        print("Invalid input. Please enter a valid integer.")
```

Example Assignment Description for Differential Testing: **"**Write a C program, **fib.c**, to calculate the nth Fibonacci number, ensuring that the program takes input from the user for the value 'n'."

Example Unit Testing Assignment

We will prepare an autograder for 'fib.py', a Python program that generates Fibonacci sequences. The program prompts users for the length of the sequence, generating it accordingly. Unit testing will verify sequence accuracy and error handling.

Step 1: Create Files for Autograder

1. setup.sh

   This script installs Python and the pip package manager. Then it uses pip to install our two external dependencies.

```
#!/usr/bin/env bash

apt-get install -y python3 python3-pip python3-dev

pip3 install -r /autograder/source/requirements.txt
```

2. run_autograder

   This script copies the student's submission to the target directory and then executes the test runner Python script.

```
#!/usr/bin/env bash

# Set up autograder files

cp /autograder/submission/fib.py /autograder/source/fib.py

cd /autograder/source

python3 run_tests.py
```

3. run_tests.py

   This python script loads and runs the tests using the JSONTestRunner class from gradescope-utils. This produces the JSON formatted output to stdout, which is then captured and uploaded by the autograder harness.

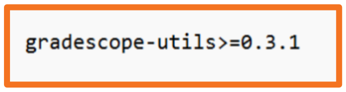
```
import unittest
from gradescope_utils.autograder_utils.json_test_runner import JSONTestRunner

if __name__ == '__main__':
    suite = unittest.defaultTestLoader.discover('tests')
    with open('/autograder/results/results.json', 'w') as f:
        JSONTestRunner(visibility='visible', stream=f).run(suite)
```

suite = unittest.defaultTestLoader.discover('tests')
JSONTestRunner().run(suite)

This will load tests from the tests/ directory in your autograder source code, loading only files starting with test by default. JSONTestRunner is included in gradescope-utils.

4. requirements.txt

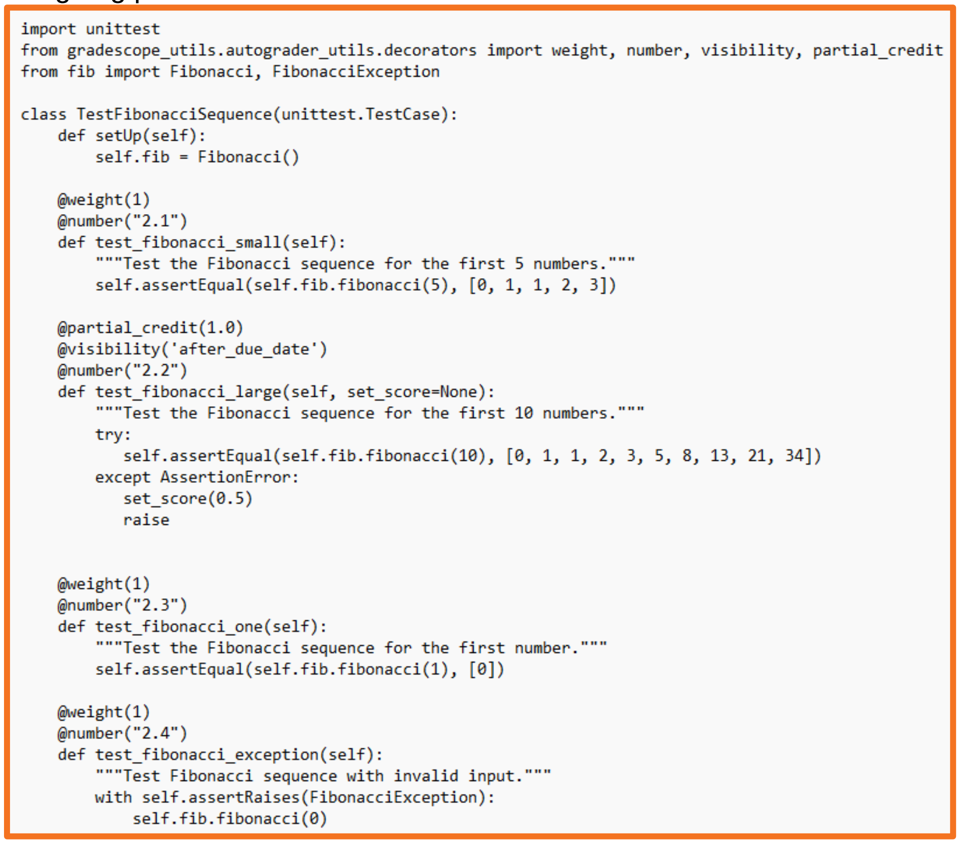
   This text file specifies the gradescope utils



```
gradescope-utils>=0.3.1
```

5. tests (folder)

   This folder contains the test cases starting with test. For our example, we will have one test case, 'test_simple.py'. The setUp method initializes a **'***Fibonacci***'** object for consistent testing. Test methods verify the **'***Fibonacci'* class's **'***fibonacci'* method with various inputs, ensuring accurate generation of the Fibonacci sequence or the raising of *FibonacciException* when appropriate. Decorators used include "Weight" for grading, "Number" for test identification, "Visibility" to control display, and "Partial_credit" for assigning partial credit on test failure.



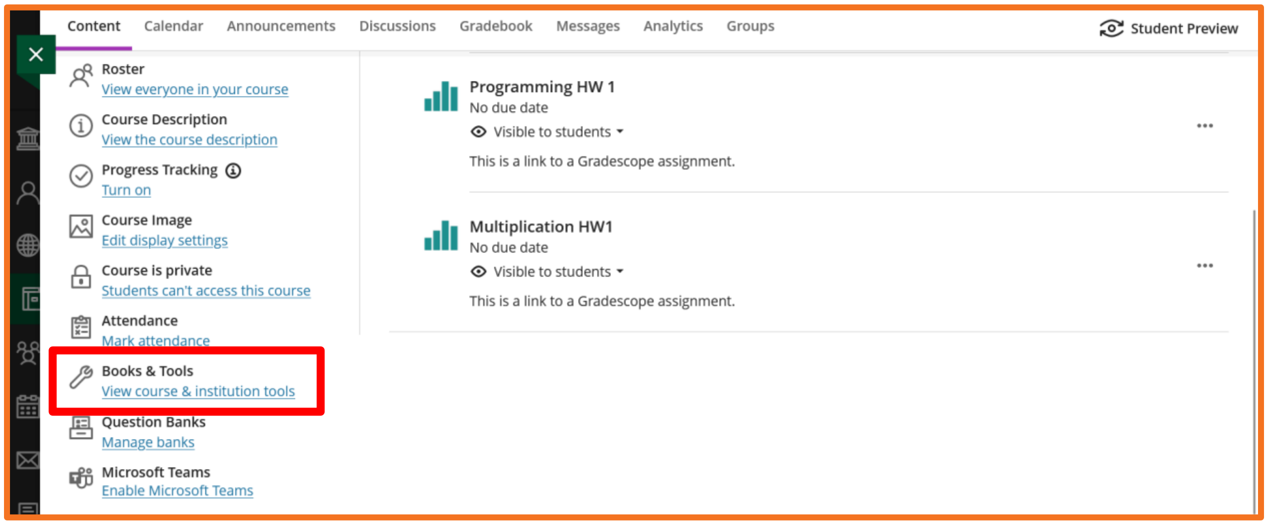
```
import unittest
from gradescope_utils.autograder_utils.decorators import weight, number, visibility, partial_credit
from fib import Fibonacci, FibonacciException

class TestFibonacciSequence(unittest.TestCase):
    def setUp(self):
        self.fib = Fibonacci()

    @weight(1)
    @number("2.1")
    def test_fibonacci_small(self):
        """Test the Fibonacci sequence for the first 5 numbers."""
        self.assertEqual(self.fib.fibonacci(5), [0, 1, 1, 2, 3])

    @partial_credit(1.0)
    @visibility('after_due_date')
    @number("2.2")
    def test_fibonacci_large(self, set_score=None):
        """Test the Fibonacci sequence for the first 10 numbers."""
        try:
           self.assertEqual(self.fib.fibonacci(10), [0, 1, 1, 2, 3, 5, 8, 13, 21, 34])
        except AssertionError:
           set_score(0.5)
           raise


    @weight(1)
    @number("2.3")
    def test_fibonacci_one(self):
        """Test the Fibonacci sequence for the first number."""
        self.assertEqual(self.fib.fibonacci(1), [0])

    @weight(1)
    @number("2.4")
    def test_fibonacci_exception(self):
        """Test Fibonacci sequence with invalid input."""
        with self.assertRaises(FibonacciException):
            self.fib.fibonacci(0)
```

Step 2: Create Autograder Zip File

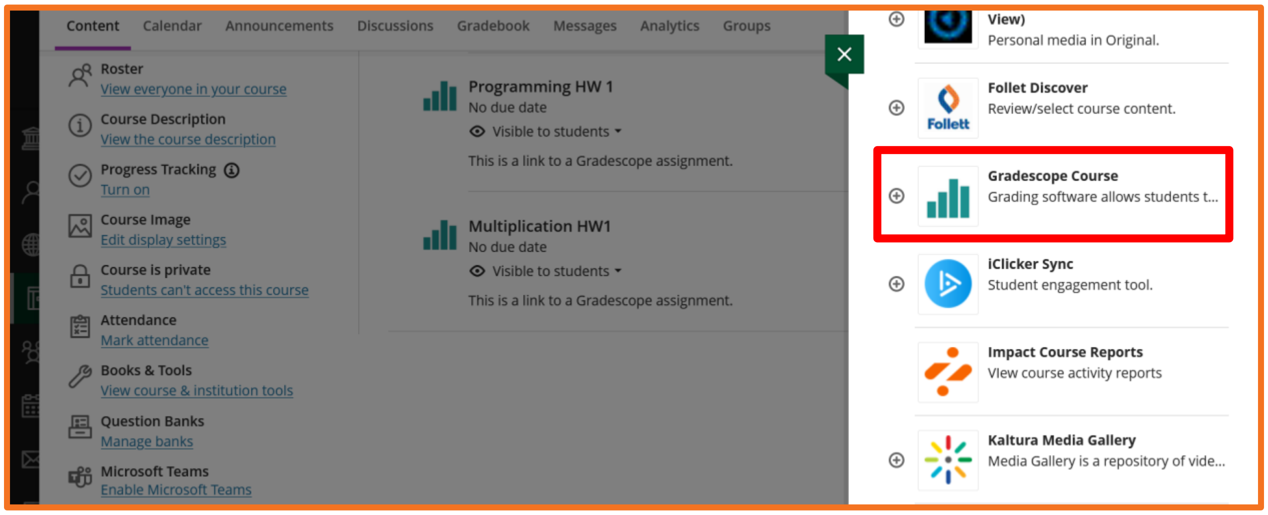
- Autograders are uploaded to Gradescope in zip format. When you are zipping up your files, make sure to zip the files, and not the folder containing the files. Ensure your zip file is named 'autograder'.

Step 3: Setup Your Blackboard Assignment

1. In your Ultra course, select **'view course & institution tools'**



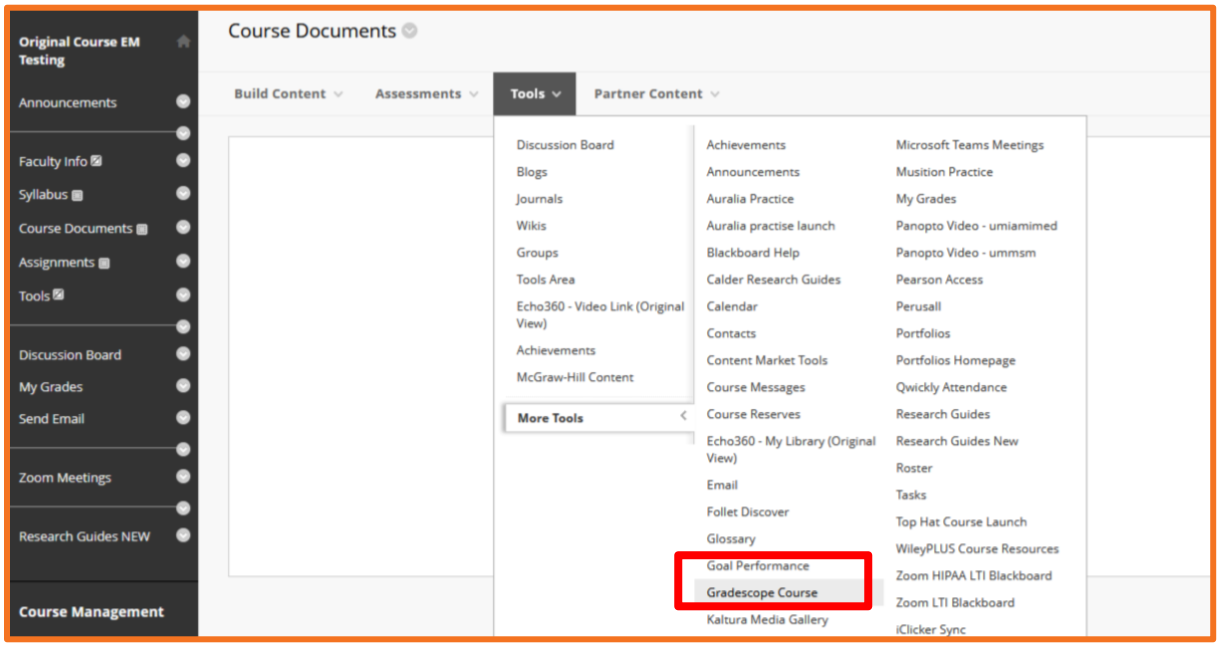
2. Select **Gradescope Course**


View)
Personal media in Original.
Follet Discover
Review/select course content.
Gradescope Course
Grading software allows students t...
iClicker Sync
Student engagement tool.
Impact Course Reports
View course activity reports
Kaltura Media Gallery
Media Gallery is a repository of vide...

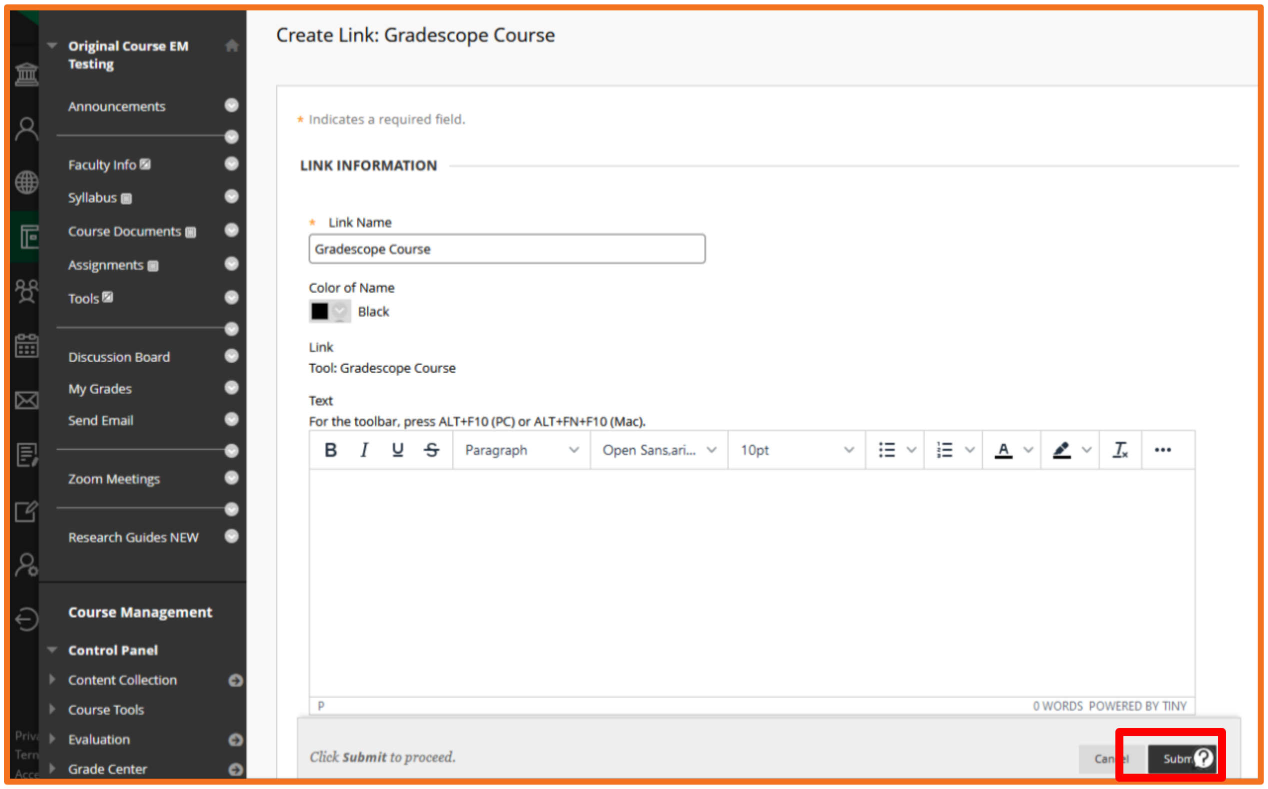
UNIVERSITY OF MIAMI
DEPARTMENT of INFORMATION TECHNOLOGY

Programming Assignment with Gradescope Autograder

3. For Original courses, select **Tools**, click **More Tools**, and choose **Gradescope Course.**



4. Click **Submit** to create a Gradescope Course link. Use this link to access Gradescope.



Create Link: Gradescope Course

* Indicates a required field.

LINK INFORMATION

* Link Name: Gradescope Course

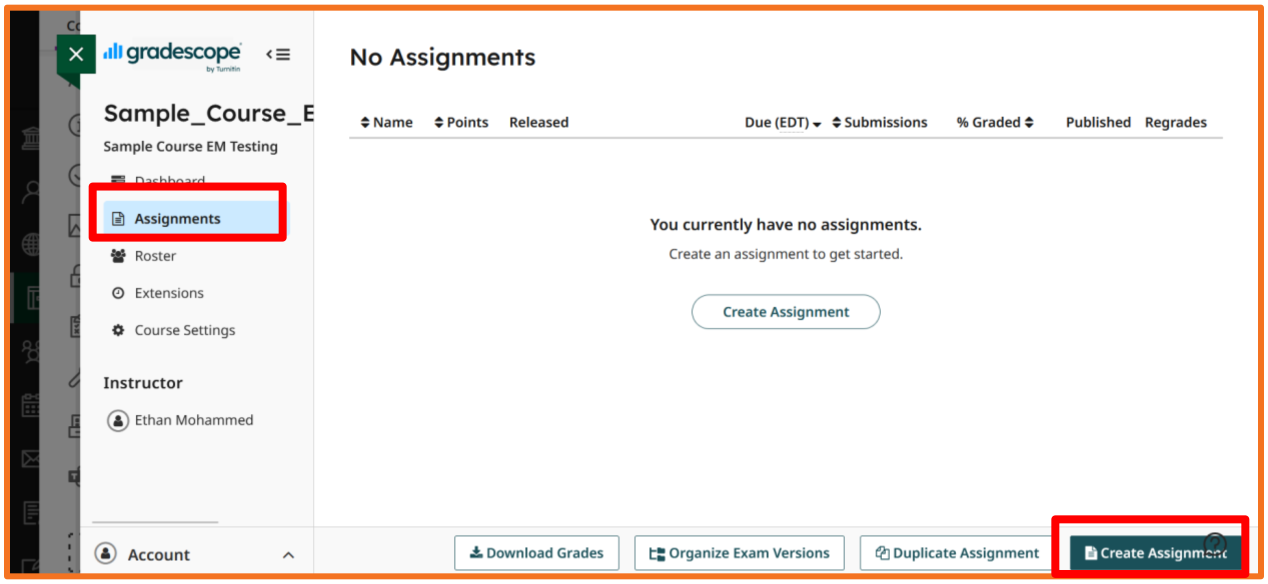
Color of Name: Black

Link
Tool: Gradescope Course

Text
For the toolbar, press ALT+F10 (PC) or ALT+FN+F10 (Mac).

*Click **Submit** to proceed.*

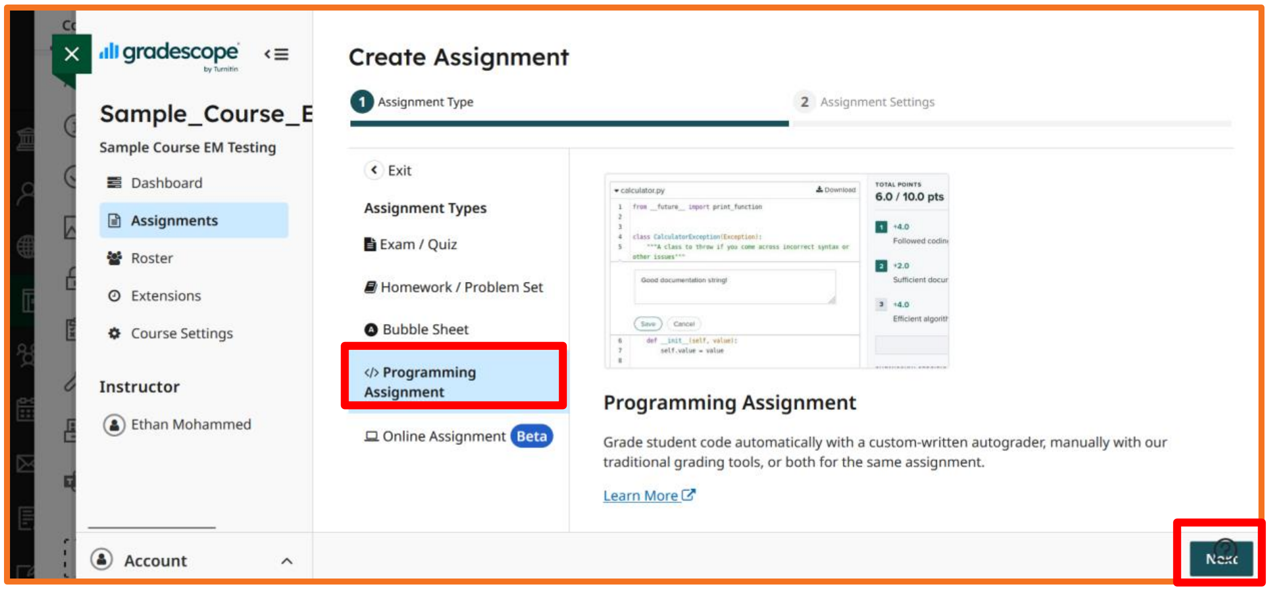
5. Setup your Gradescope course, select **Assignments** in the left navigation, and click **Create Assignment**



You currently have no assignments.

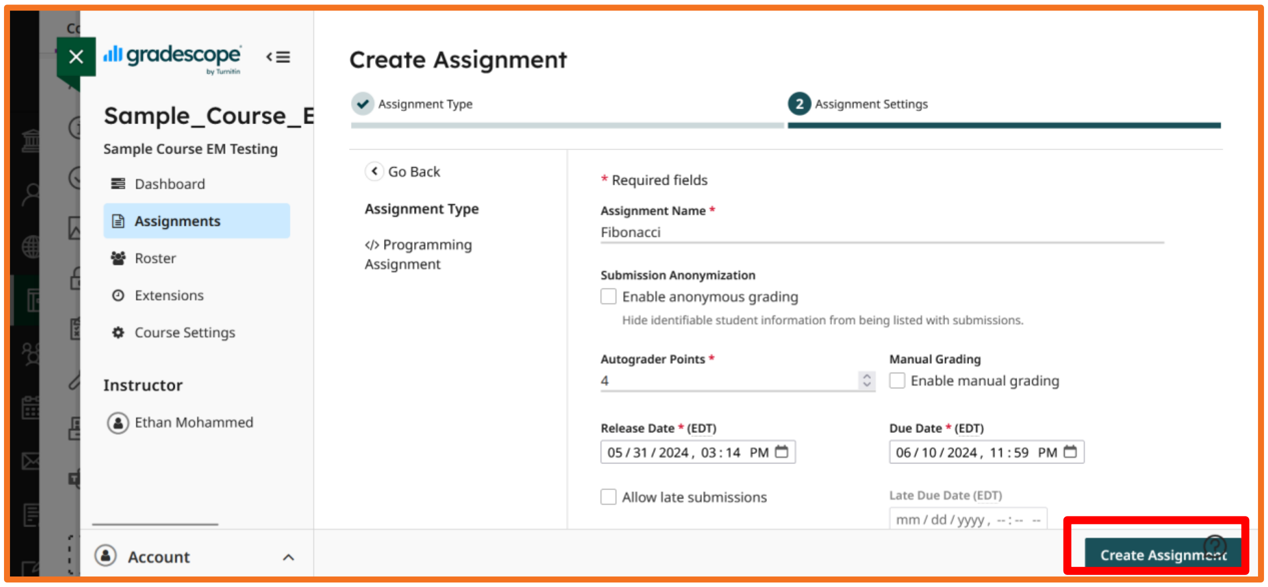
Create an assignment to get started.

6. Select **Programming Assignment**, then click **Next**



Grade student code automatically with a custom-written autograder, manually with our traditional grading tools, or both for the same assignment.

7. Provide the assignment name, release date, due date, and autograder points. Autograder points are the total of test case weights, which in this case is 4.



Create Assignment

* Required fields

Assignment Name *: Fibonacci

Submission Anonymization: Enable anonymous grading — Hide identifiable student information from being listed with submissions.

Autograder Points *: 4

Manual Grading: Enable manual grading

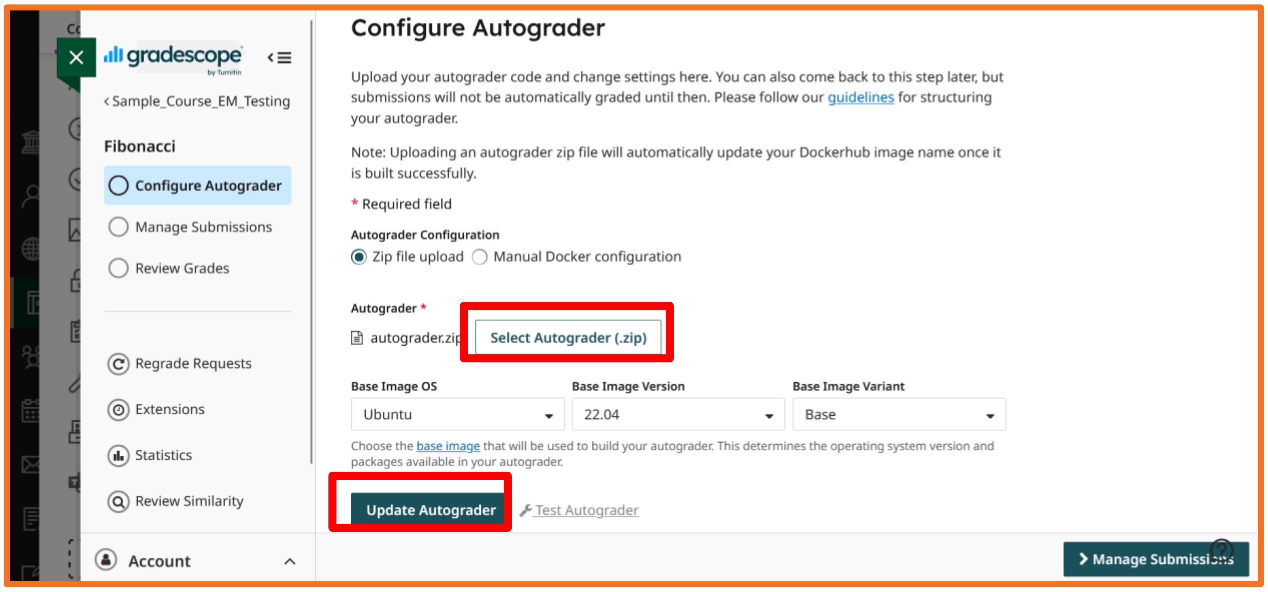
Release Date * (EDT): 05 / 31 / 2024 , 03 : 14 PM

Due Date * (EDT): 06 / 10 / 2024 , 11 : 59 PM

Allow late submissions

Late Due Date (EDT): mm / dd / yyyy , -- : -- --

8. Select your autograder zip file, then click **Update 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 *: autograder.zip

Base Image OS: Ubuntu

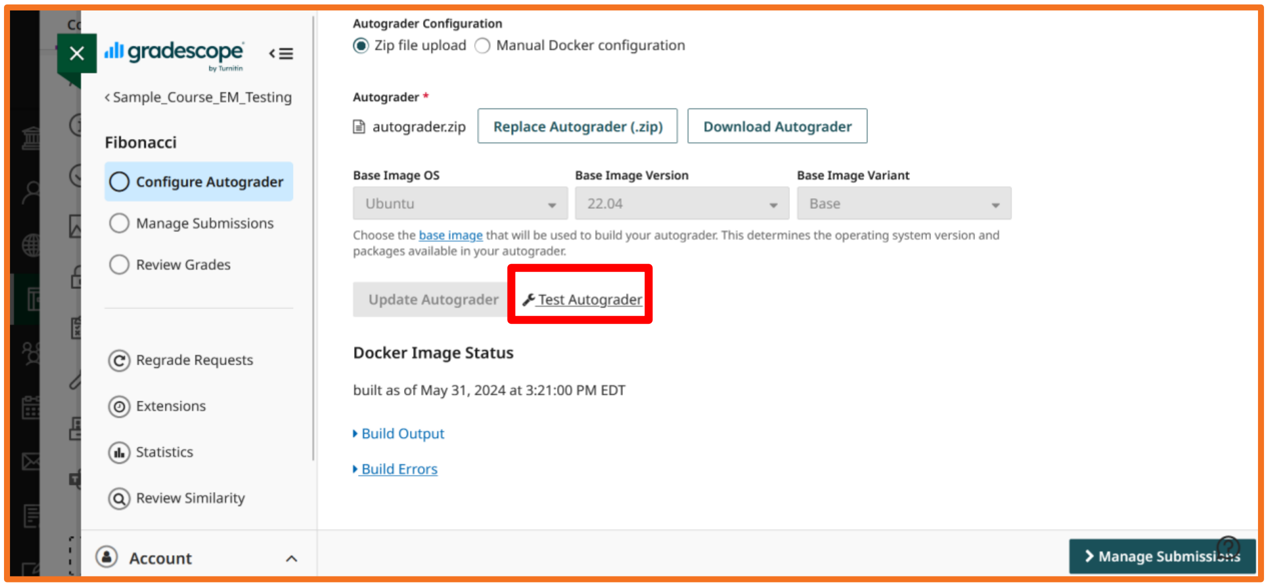
Base Image Version: 22.04

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.

Programming Assignment with Gradescope Autograder

9. After your autograder has been successfully built, click **'Test Autograder'**.



Autograder Configuration
Zip file upload [✓] Manual Docker configuration

Autograder *
autograder.zip

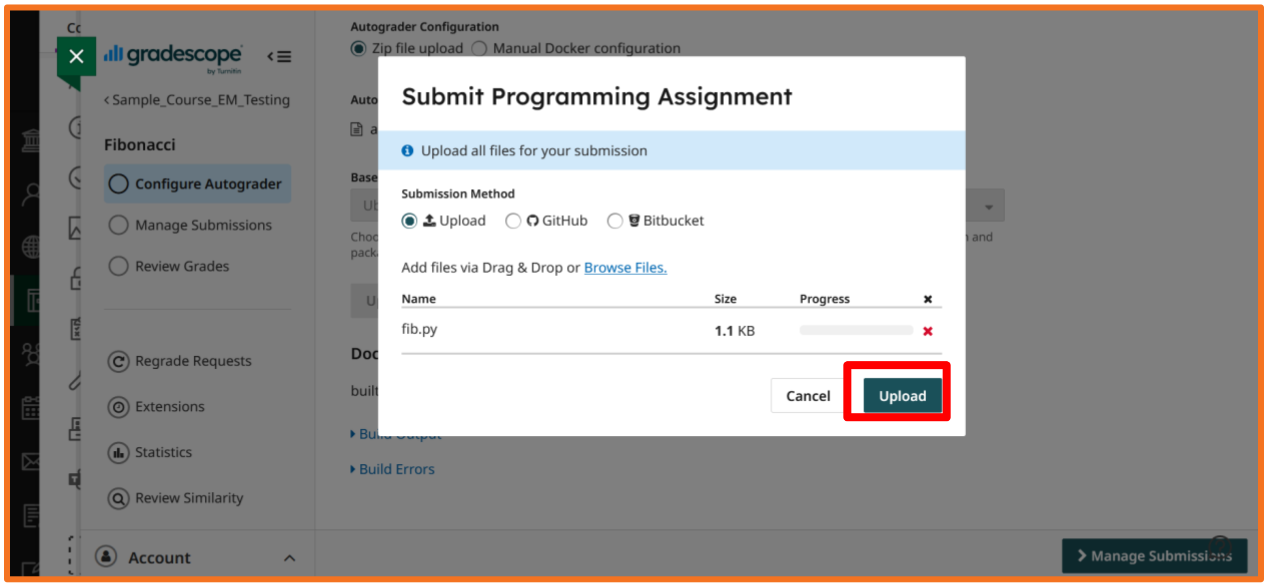
Base Image OS	Base Image Version	Base Image Variant
Ubuntu	22.04	Base

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

Docker Image Status

built as of May 31, 2024 at 3:21:00 PM EDT

10. Upload the solution 'fib.py' to the autograder, then click **Upload**



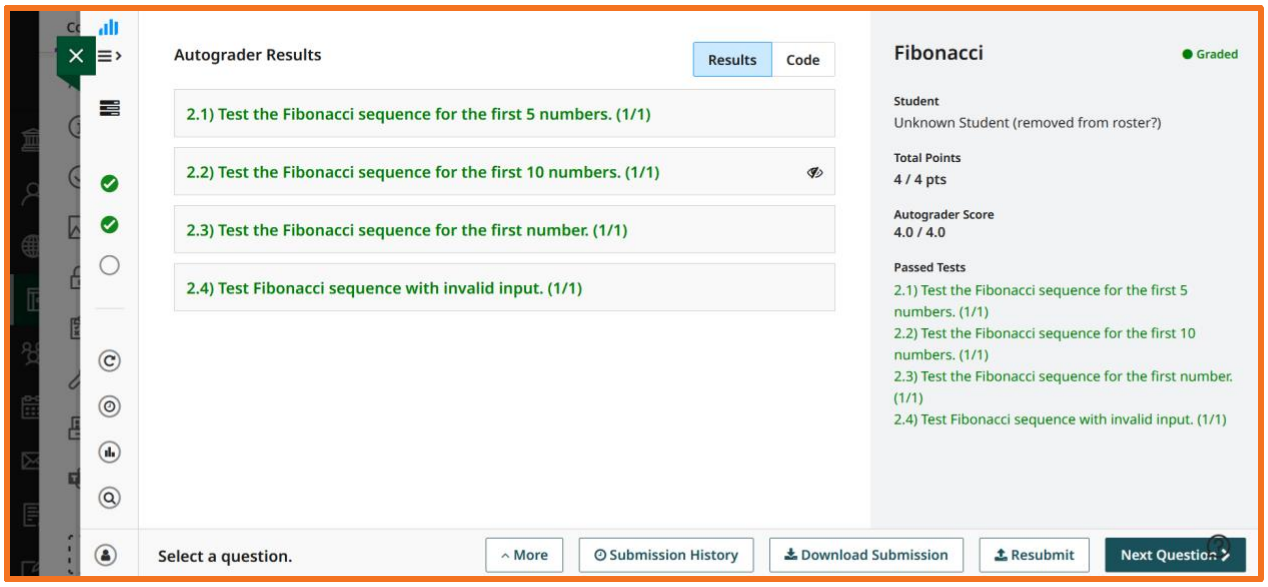
Upload all files for your submission

Submission Method
[✓ Upload] GitHub Bitbucket

Add files via Drag & Drop or Browse Files.

Name	Size	Progress
fib.py	1.1 KB	

11. View results.



2.1) Test the Fibonacci sequence for the first 5 numbers. (1/1)
2.2) Test the Fibonacci sequence for the first 10 numbers. (1/1)
2.3) Test the Fibonacci sequence for the first number. (1/1)
2.4) Test Fibonacci sequence with invalid input. (1/1)

Fibonacci
Graded

Student
Unknown Student (removed from roster?)

Total Points
4 / 4 pts

Autograder Score
4.0 / 4.0

Passed Tests
2.1) Test the Fibonacci sequence for the first 5 numbers. (1/1)
2.2) Test the Fibonacci sequence for the first 10 numbers. (1/1)
2.3) Test the Fibonacci sequence for the first number. (1/1)
2.4) Test Fibonacci sequence with invalid input. (1/1)

12. Deploy to Blackboard using the instructions found here ‘Syncing your roster.’

Example Differential Testing Assignment

We will prepare an autograder for 'fib.c', a C program that calculates the nth Fibonacci number. The idea is that you can compile the student's code and then execute it in a subprocess using Python. Then you can communicate with the subprocess by providing arguments via the command line, or via standard input, and read standard output to see what the program produced.

Step 1: Create files for Autograder:

1. setup.sh

   This script installs Python and the pip package manager. Then it uses pip to install our two external dependencies.

```
#!/usr/bin/env bash

apt-get install -y python3 python3-pip python3-dev

pip3 install -r /autograder/source/requirements.txt
```

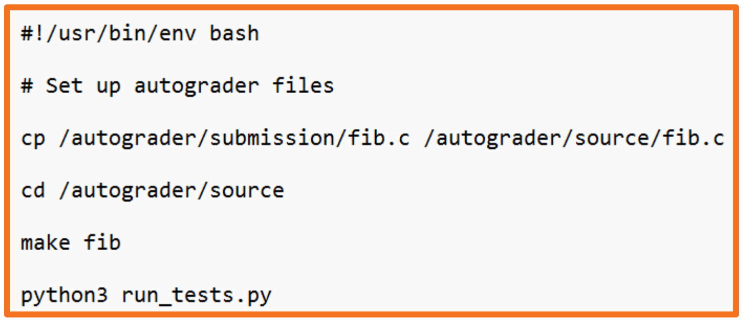
2. run_tests.py

   This python script loads and runs the tests using the JSONTestRunner class from gradescope-utils. This produces the JSON formatted output to stdout, which is then captured and uploaded by the autograder harness.

```
import unittest
from gradescope_utils.autograder_utils.json_test_runner import JSONTestRunner

if __name__ == '__main__':
    suite = unittest.defaultTestLoader.discover('tests')
    with open('/autograder/results/results.json', 'w') as f:
        JSONTestRunner(visibility='visible', stream=f).run(suite)
```

3. run_autograder

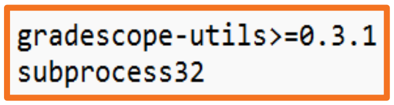
   This script copies the student's 'fib.c' submission to a designated directory, compiles it into an executable, and then runs tests using a Python script named 'run_tests.py'.



```
#!/usr/bin/env bash

# Set up autograder files

cp /autograder/submission/fib.c /autograder/source/fib.c

cd /autograder/source

make fib

python3 run_tests.py
```

4. requirements.txt

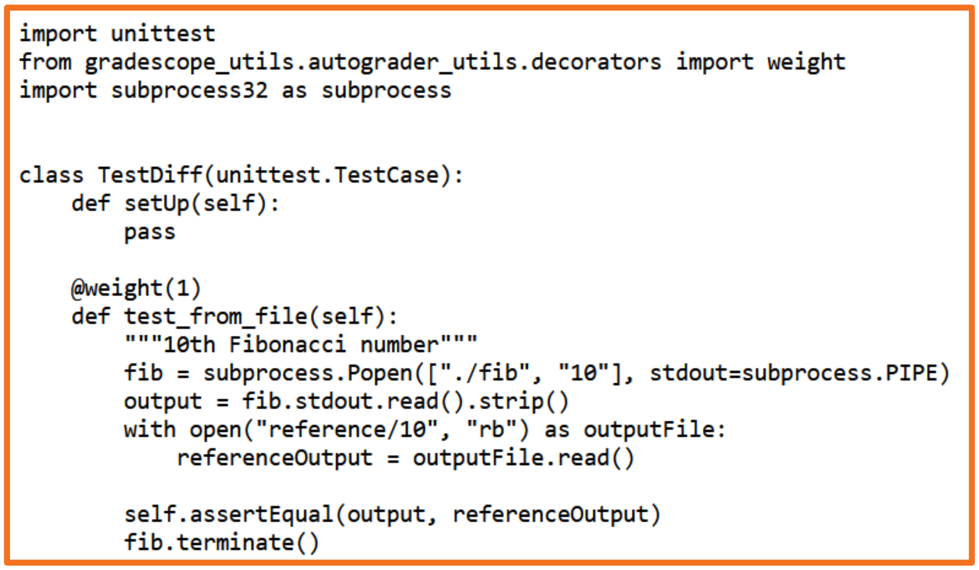
   This text file specifies the gradescope utils and subprocess



```
gradescope-utils>=0.3.1
subprocess32
```

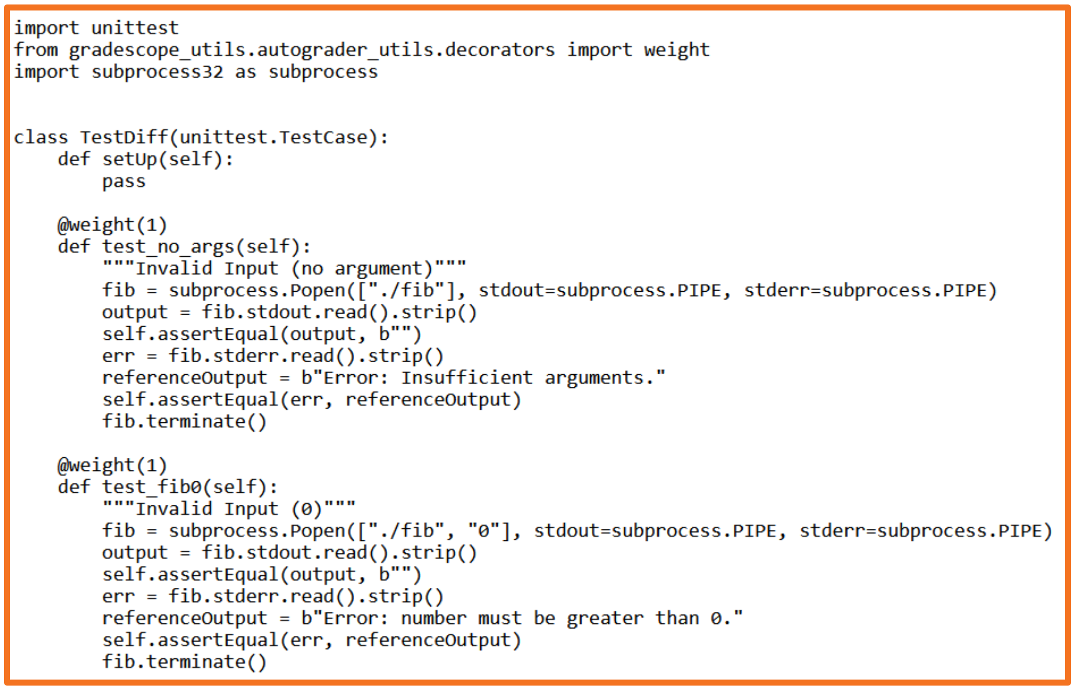
5. tests (folder)

   This folder contains the test cases starting with test. For our example, we will have two test cases ‘test_from_file.py’ and ‘test_subprocess.py’. test_from_file.py executes the 'fib' program with an argument of 10 to generate the 10th Fibonacci number. It compares the output with a reference output stored in a file named 'reference/10', asserting their equality to validate correctness



```
import unittest
from gradescope_utils.autograder_utils.decorators import weight
import subprocess32 as subprocess


class TestDiff(unittest.TestCase):
    def setUp(self):
        pass

    @weight(1)
    def test_from_file(self):
        """10th Fibonacci number"""
        fib = subprocess.Popen(["./fib", "10"], stdout=subprocess.PIPE)
        output = fib.stdout.read().strip()
        with open("reference/10", "rb") as outputFile:
            referenceOutput = outputFile.read()

        self.assertEqual(output, referenceOutput)
        fib.terminate()
```

For test_subprocess.py, each test method checks different scenarios, such as invalid inputs and specific Fibonacci number calculations, by running the 'fib' program with subprocess and comparing its output with predefined reference outputs. The '@weight' decorator assigns a weight of 1 to each test case for grading purposes.



```
import unittest
from gradescope_utils.autograder_utils.decorators import weight
import subprocess32 as subprocess


class TestDiff(unittest.TestCase):
    def setUp(self):
        pass

    @weight(1)
    def test_no_args(self):
        """Invalid Input (no argument)"""
        fib = subprocess.Popen(["./fib"], stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        output = fib.stdout.read().strip()
        self.assertEqual(output, b"")
        err = fib.stderr.read().strip()
        referenceOutput = b"Error: Insufficient arguments."
        self.assertEqual(err, referenceOutput)
        fib.terminate()

    @weight(1)
    def test_fib0(self):
        """Invalid Input (0)"""
        fib = subprocess.Popen(["./fib", "0"], stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        output = fib.stdout.read().strip()
        self.assertEqual(output, b"")
        err = fib.stderr.read().strip()
        referenceOutput = b"Error: number must be greater than 0."
        self.assertEqual(err, referenceOutput)
        fib.terminate()
```

6. reference (folder)

   This folder contains a file named ‘10’, which contains ‘55’. This file is used for the test_from_file.py where the referenceOutput is found in reference/10.

Step 2: Create Autograder Zip File

- Autograders are uploaded to Gradescope in zip format. When you are zipping up your files, make sure to zip the files, and not the folder containing the files. Ensure your zip file is named ‘autograder’.

Step 3: Setup Your Blackboard Assignment

After following the steps described earlier, test the Autograder with the solutions file 'fib.c':