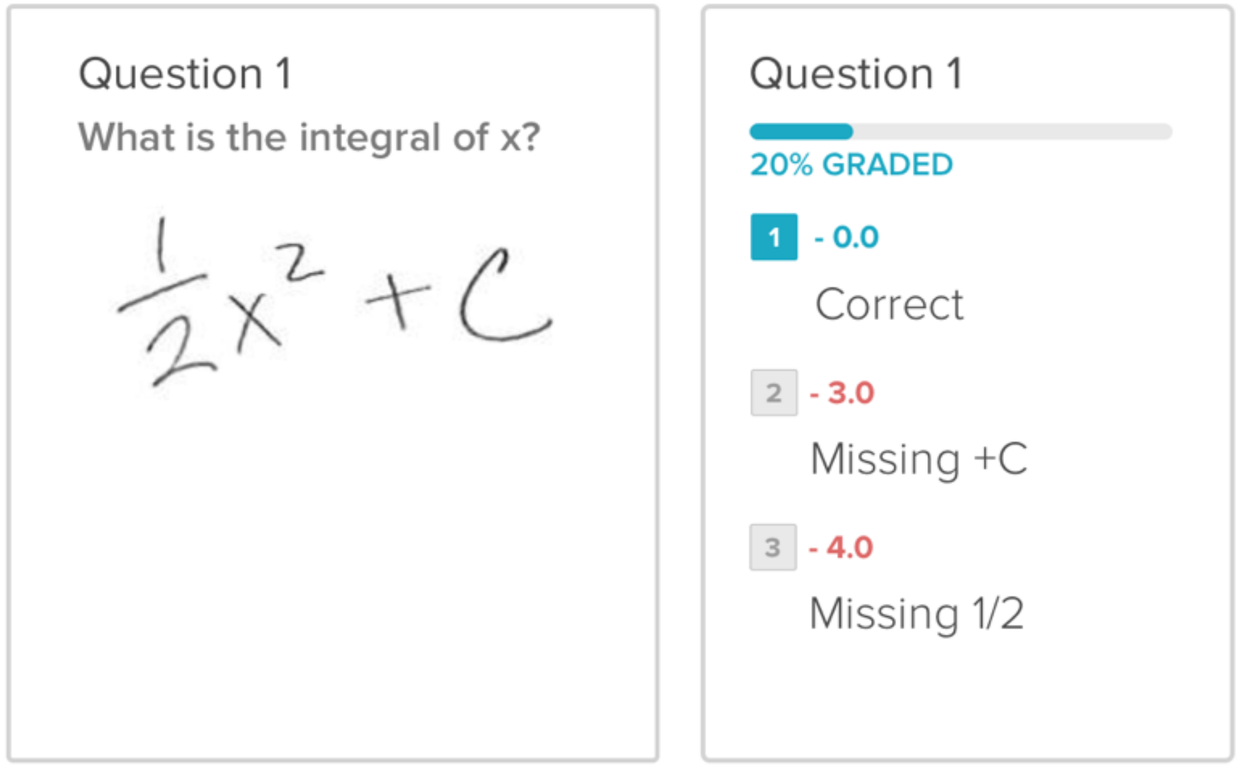
gradescope

SIGCSE 2016 Supporter Session

- Single place to grade all of your students' work.
  - Since 2014: Exams, homework (handwritten or typed)
  - Now: help autograde programming assignments



outline

- How do you write an autograder?
  - Example implementation of simple autograder
- How do you let students use your autograder?
  - Expose autograder to students via Gradescope

Goal: take student code, give students feedback and grade.

```
def fib(n):
    if x <= 1:
        return 1
    else:
        return fib(x-1) + fib(x-2)
```

Correct (5/5)

- Many ways to do it. See [1] for survey.
- Most popular: unit test based and diff based
- unit test based
  - test small components of student code
  - compare student output with expected output
- diff based
  - diff output of reference solution with student submission
  - fail if any differences

```
def test_fib_5(self):
    self.assertEqual(fib(5), 5)

def test_fib_6(self):
    self.assertEqual(fib(6), 8)
```

```
--- student_output
+++ reference_output
@@ -3,4 +3,4 @@
 2
 3
 5
-7
+8
```

[1] Ihantola, P., Ahoniemi, T., Karavirta, V. and Seppälä, O., 2010, October. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (pp. 86-93). ACM.

Unit Tests

- Pros
  - used in industry
  - easy to understand
  - gives reasonable feedback (usually)
- Cons
  - tests are language specific
  - requires knowledge of unit testing framework

```
def test_fib_5(self):
    self.assertEqual(fib(5), 5)

def test_fib_6(self):
    self.assertEqual(fib(6), 8)
```

Diff

- Pros
  - language independent
  - easy to write
- Cons
  - brittle — without extra work, whitespace and case sensitive
  - requires extra work to give partial credit

```
--- student_output
+++ reference_output
@@ -3,4 +3,4 @@
 2
 3
 5
-7
+8
```

Project: build a calculator that reads and evaluates expressions from standard input.

- Requirements
  - The calculator should handle the 4 basic operations, +, -, *, /, with operator precedence
  - It should handle parentheses and negative numbers
  - If the user types 'quit', exit the program

example: calculator

```
bash-3.2$ python calculator.py
> 4
```

Project: write calculator that runs until user enters ‘quit’.

```
class Calculator(object):

    def read(self):
        """Read input from stdin"""
        return raw_input('> ')

    def eval(self, string):
        """Evaluates an infix arithmetic expression"""
        # TODO: Implement me
        pass

    def loop(self):
        """Read a line of input, evaluate it, and print it.
        Repeat until the user types 'quit'."""
        line = self.read()
        # TODO: Implement the loop
        pass

if __name__ == '__main__':
    calc = Calculator()
    calc.loop()
```

example: calculator

```
import unittest
from calculator import Calculator

class CalculatorTestCase(unittest.TestCase):
    def setUp(self):
        self.calc = Calculator()

    def test_eval_add(self):
        """Test evaluating 1 + 1"""
        val = self.calc.eval("1 + 1")
        self.assertEqual(val, 2)

    def test_eval_parens(self):
        """Test evaluating (1 + 1) * 4"""
        val = self.calc.eval("(1 + 1) * 4")
        self.assertEqual(val, 8)

    def test_eval_precedence(self):
        """Test evaluating 1 + 1 * 8"""
        val = self.calc.eval("1 + 1 * 8")
        self.assertEqual(val, 9)
```

- We have our autograder. Now what?
- Want to let students submit and run autograder to give feedback
- How do you let students use your autograder?

- Students submit
  - email, web interface, LMS, git, etc.
- Collect student submissions & run autograder on them
  - Real time, batch after deadline, “human autograding”
    - need infrastructure for real time. TA becomes sysadmin
- Provide feedback to students
  - email, web interface, LMS

- Gradescope provides all needed infrastructure
  - Students submit, we run your autograders, we give students your feedback
- Support all languages — you install whatever you want
- Handle security & isolation concerns
- No need to worry about load spikes around deadlines

Gradescope interface

Need three things, all in a zip file.

1. setup.sh
    - install any languages and libraries you need

2.run_autograder
    - a script that runs your test cases and outputs a JSON document

3. any support code (including test cases)

setup.sh

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

apt-get install -y python python-pip python-dev

pip install subprocess32 gradescope-utils
```

Install Python. Install dependencies.

run_autograder interface

- We put your code in /autograder/source
- We put student code in /autograder/submission
- run_autograder outputs JSON document to file:
  - /autograder/results/results.json

run_autograder: JSON Schema

```
{
  "output": "Optional text that shows up at the top of the results.",
  "tests":
    [
        {
            "score": 2.0,
            "max_score": 2.0,
            "name": "Test #1",
            "output": "Formatted multiline string."
        },
        {
            "score": 1.0,
            "max_score": 2.0,
            "name": "Test #2",
            "output": "Formatted multiline string."
        },
    ]
}
```

How do we get this JSON structure?

run_autograder: JSON Schema

```python
from gradescope_utils.autograder_utils.decorators import weight

@weight(2)
def test_eval_parens(self):
    """Test evaluating (1 + 1) * 4"""
    val = self.calc.eval("(1 + 1) * 4")
    self.assertEqual(val, 8)

@weight(2)
def test_eval_precedence(self):
    """Test evaluating 1 + 1 * 8"""
    val = self.calc.eval("1 + 1 * 8")
    self.assertEqual(val, 9)
```

Write some code to transform the output into JSON.
For Python, use our utilities.

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

# Copy student code into place
cp /autograder/submission/calculator.py /autograder/source/calculator.py

cd /autograder/source

# Run autograder
python run_tests.py > /autograder/results/results.json
```

```
bash-3.2:~/autograder$ ls
run_autograder    run_tests.py    setup.sh    tests

bash-3.2:~/autograder$ ls tests
__init__.py    test_complex.py    test_integration.py
test_simple.py test_unknown.py
```