CIS129 FINAL

PRE-GRADING SOFTWARE

by Amy Cardona

AUTOMATED GRADING

Automated grading uses software to quickly evaluate student work like tests, essays, or code, providing instant feedback and saving instructors time, and has been evolving since the 1960s.

Automatic Machine Grading Programs

George E. Forsythe (Stanford University, 1964)

Developed for programming/numerical analysis courses at Stanford.

Inspired by TEACH routine from Carnegie Tech.

Programs (GRADER1, GRADER2...) automated:

Data generation

Answer checking

Time tracking

Grade logging

Built in BALGOL; integrated via machine-language procedures.

Allowed custom graders per assignment with minimal student effort.

Enabled scalable grading for large classes.

Later systems lacked flexibility, limiting adoption.

Forsythe recommends grading programs but advises system compatibility checks.

THE PROBLEM

Course equivalency evaluation would use similar unit testing and used similar ideas via checking off criteria

Originak Idea: Course Evaluation between colleges

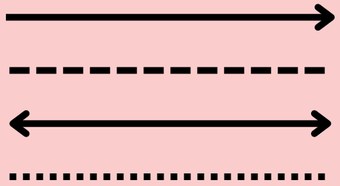
Many students also don't fully understand rubrics or how their work is scored.

- Lack of clarity in rubrics and expectations.
- No immediate way to estimate assignment quality.
- Delayed feedback can lead to repeated mistakes in future assignments.
- Provides instant feedback, helping students learn and improve faster

Based on automated grading programs which:

- Speed up grading for large classes and ensures consistent, unbiased results.
- Reduces instructor workload by automating assessment and grade recording.

Not knowing their grade can lead students to feel anxious, lack direction for improvement, and lose motivation, ultimately hindering their learning and progress.



Why It Matters: Students often submit work without knowing how well they met expectations. Waiting for feedback can cause stress and delay improvement

CURRENT ALTERNATIVES

Codewars | Gradescope | GitHub Classroom

Codewars is more of a coding practice however uses a self grading system to gain mastery in desired coding subject, codewars also allows for integration with current github account which is pretty cool

Gradescope by Turnit-in offers an auto grading system however teachers must upload their own zip file of code with their adjusted rubric requirements

Grammarly/QuillBot: Focus on grammar, not full assignment evaluation.

Google Classroom Rubrics: For teachers, not students.

Turnitin: Gives feedback, but only after submission.

Gap: No tool offers student-first grading before submitting work..

KEY FEATURES

Goal: Let students select an assignment type, review the rubric, self-score (or use AI suggestions), and get an estimated grade.

Inputs: Assignment type, uploaded work or written input, rubric selection.

Outputs: Estimated grade, score breakdown, revision tips.

Rubric templates for essays, lab reports, presentations, coding projects, and research papers.

Self-scoring/ Automated grading

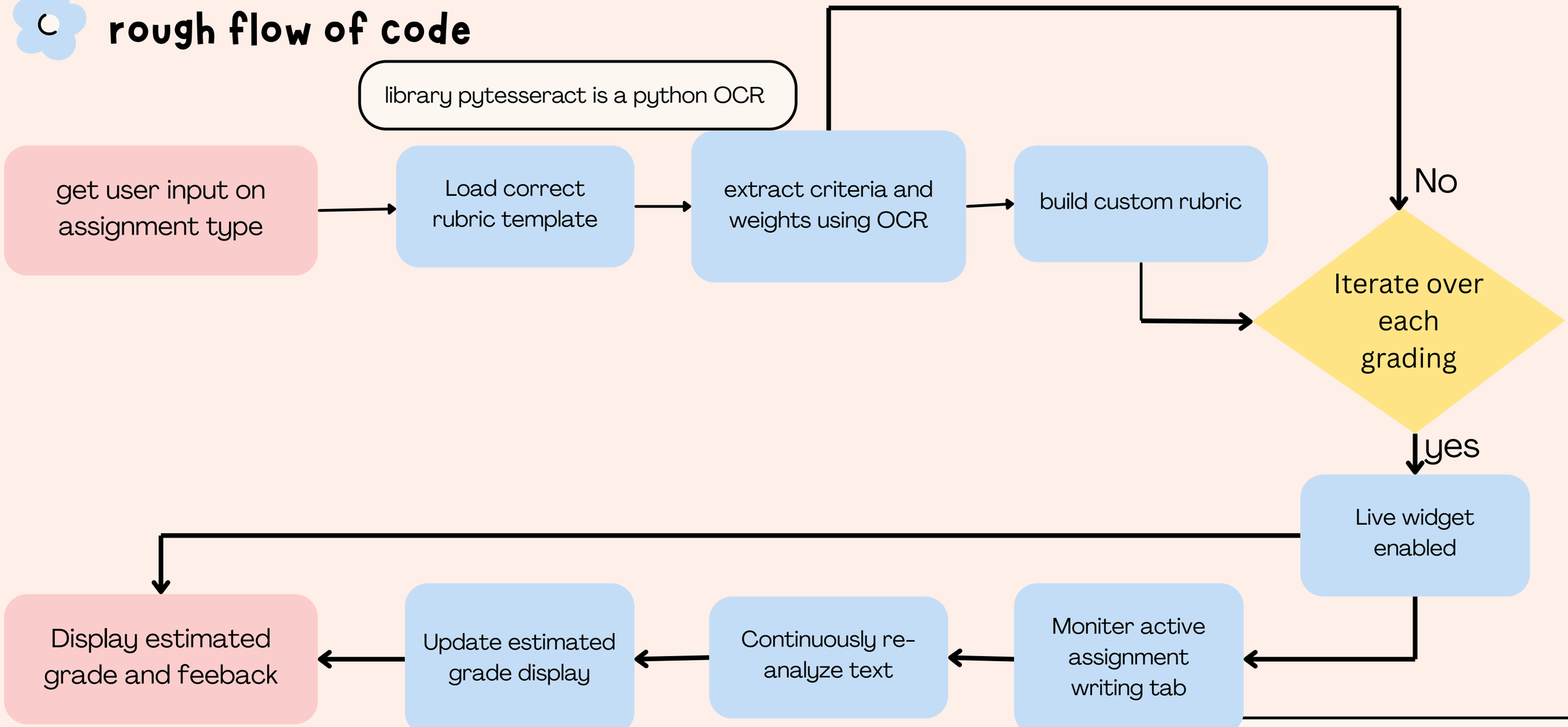
Upload teacher's rubric as a PDF and auto-extract criteria.

Revision suggestions per rubric category.

Exportable feedback PDF.

Live Grade Widget: An Optional feature that monitors your assignment tab and gives real-time updates on your current estimated grade as you write. It can be turned on/off by the user.

rough flow of code



Optical Character Recognition (OCR) in Python involves using libraries and tools to extract text from images. It is a technology that converts images of text into machine-readable text.

MORE ON PYTESSERACT

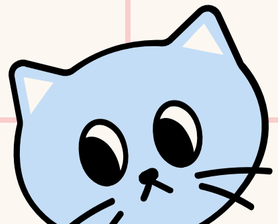
```
from PIL import Image
import pytesseract
import os

# Set the path to the Tesseract executable (only needed if it's not in PATH)
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

# Path to uploaded image (update this based on your upload logic)
uploaded_image_path = 'uploaded_image.png'

# Ensure the file exists
if os.path.exists(uploaded_image_path):
    # Open and process the uploaded image
    img = Image.open(uploaded_image_path)
    text = pytesseract.image_to_string(img)
    print("Extracted Text:\n", text)
else:
    print("Error: Uploaded image not found.")
```

Pytesseract helps my autograding software extract grading criteria directly from uploaded rubric images or PDFs using OCR. It streamlines rubric creation, supports printed and handwritten formats, and reduces the need for manual input.



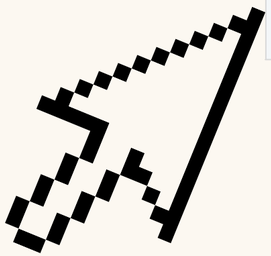
UNIT-TESTING

Unit Testing is the first level of software testing where the smallest testable parts of software are tested. This is used to validate that each software unit performs as designed.

- Ensures Reliability of Core Functions
- Verifies that each part (e.g. rubric parsing, score calculation, live feedback logic) works as intended.
- Prevents regressions when you update or expand the system.
- Catches Edge Cases
- Tests how the system handles:
  - Missing or malformed rubrics
  - Unscored criteria
  - Unexpected input (e.g. non-text PDFs, odd formatting, empty assignments)
- Supports Refactoring and Scaling
- Want to swap in a new OCR library or scoring model? Unit tests make sure old functionality still works.
- You'll know immediately which part of the system broke and why.
- Makes debugging faster and safer.
- If you're autograding other people's work, errors can have real consequences (unfair scores, missed feedback). Unit tests reduce that risk.

FUNCTIONS AND POTENTIAL USAGE

Component	Test Examples
load_corresponding_rubric()	Loads correct rubric for essay, lab, etc.
extract_criteria_and_weights()	Handles real rubric files, bad scans, missing weights
build_custom_rubric()	Creates correct rubric structure from extracted data
get_user_score_or_auto_score()	AI scores or manual scores match expected format/range
calculate_total_score()	Total = weighted sum; edge cases (zero weights, partial input)
live_widget_enabled()	Triggers correctly; updates scores in real time



PSEUDOCODE DRAFT

```
# 1. Get user input on assignment type
assignment_type = input("Select assignment type (essay, lab, project,
etc.):")

# 2. Load appropriate rubric template
rubric = load_corresponding_rubric(assignment_type)

# 3. Optional: Handle custom rubric upload
if uploaded_rubric_file:
    # Extract criteria and weights using OCR from uploaded rubric
    criteria, weights =
extract_criteria_and_weights_OCR(uploaded_rubric_file)
    rubric = build_custom_rubric(criteria, weights)

# 4. Iterate over each grading criterion
for criterion in rubric:
    display(criterion.description)
    # Optionally let the user score or use AI evaluation
    score = get_user_score_or_auto_score(criterion, user_input)
```

ROUGH OVERVIEW

```
# 4. Iterate over each grading criterion
for criterion in rubric:
    display(criterion.description)
    # Optionally let the user score or use AI evaluation
    score = get_user_score_or_auto_score(criterion, user_input)
    criterion.score = score

# 5. Optional: Enable Live Grade Widget
if live_widget_enabled:
    # Monitor user's assignment editing tab
    while editing_assignment_tab_active():
        assignment_text = capture_current_assignment_text()
        updated_scores = reanalyze_text(assignment_text, rubric)
        update_estimated_grade_display(updated_scores)

# 6. Final calculation and feedback
total_score = calculate_total_score(rubric)
feedback = generate_feedback(rubric)
display(total_score, feedback)
```

REFLECTION

- Helps students write with intent.
- Teaches students to understand rubrics.
- Encourages revision and growth.
- Can be used by tutors, study groups, and writing centers.

I thought I would need to look into java for web developement and was shocked to find out how much can be done with python and its libraries.

I think it would be a fun summer project to try and make this code

Can rubric extraction from PDF be made accurate across all formats? Will the live widget be too distracting or helpful for different writing styles?

CITATIONS

- “Grammarly: Free Writing AI Assistance.” Grammarly, www.grammarly.com.
- “Use Rubrics for Assignments.” Google for Education Help, Google, support.google.com/edu/classroom/answer/9335069?hl=en.
- “Turnitin.” Turnitin, www.turnitin.com.
- “Writing Center.” The University of North Carolina at Chapel Hill, writingcenter.unc.edu.
- “Use Autograding.” GitHub Docs, GitHub, https://docs.github.com/en/education/manage-coursework-with-github-classroom/teach-with-github-classroom/use-autograding.
- “Acardona5’s Profile.” Codewars, https://www.codewars.com/users/acardona5.
- “Specs — Gradescope Autograders Documentation.” Gradescope, https://gradescope-autograders.readthedocs.io/en/latest/specs/.
- “Python Support — Gradescope Autograders Documentation.” Gradescope, https://gradescope-autograders.readthedocs.io/en/latest/python/.
- Gradescope Python Autograder Sample Project. GitHub, https://github.com/gradescope/autograder_samples/tree/master/python/src.
- Calculator.py - Gradescope Python Example. GitHub, https://raw.githubusercontent.com/gradescope/autograder_samples/master/python/src/solution/calculator.py.
- Forsythe, George E. Automatic Machine Grading Programs. Stanford University, 1964. Association for Computing Machinery, https://doi.org/10.1145/800257.808930.

THAT'S ALL
THANKYOU