Automated and Configurable Programming Project Checking with Chasten

Daniel Bekele, Jaclyn Pham, and Gregory M. Kapfhammer

May 15, 2025

PyCon Education Summit 2025

What Problem Are We Solving?

- Students may struggle to write **efficient, readable code**
- Manual project review is **time-consuming** and **error-prone**
- Many courses face these challenges:
  - Data structures
  - Algorithm analysis
  - Software engineering
- Existing tools focus on **style**, not **semantic structure**
- Regex is **brittle**, and AST tools are **hard to prototype**

💡 **Project Goal**: Chasten enables **scalable** and **structure-aware** feedback that effectively supports both instructors and students

Avoid Time Complexity of $O(n^2)$

```python
# O(n) is acceptable
seen = set()
for item in items:
    if item in seen:
        return True
    seen.add(item)
```

```python
# O(n²) is not okay
for i in range(len(items)):
    for j in range(len(items)):
        if i != j
         and items[i] == items[j]:
            return True
```

- 💡 **Goal**: Automatically scan the source code that students submit to confirm that there are no inefficient looping constructs
- ⚠ **Challenge**: Linters like Ruff and Pylint don't have rules to detect nested control structures that either are or are not acceptable
- ⚙ **Build**: An extensible tool allowing instructors to scan for arbitrary code patterns without detailed AST knowledge

Chasten to the Rescue!

- ⚙️ Uses XPath to search Python's AST
- 📄 Rules written in simple YAML
- 🏗️ Structure-first, not just style
- 🗄️ Outputs to JSON, CSV, or SQLite

💡 **Result**: Instructors define checks once and use Chasten to easily apply them at scale across all student submissions

```yaml
- name: "nested-loops"
  code: "PERF001"
  pattern: "//For[descendant::For]"
  description: "Detects doubly nested for-loops (e.g., O(n²))"
```

Let’s Run Chasten!

Install the Tool

```
pipx install chasten  # Install Chasten in venv
pipx list             # Confirm installation
chasten --help        # View available commands
```

Run Chasten

```
chasten analyze time-complexity-lab \
        --config chasten-configuration \
        --search-path time-complexity-lab \
        --save-directory time-complexity-results \
        --save
```

- Save results to a **JSON file** and produce **console output**
- Configure the **return code** for different **detection goals**

Results from Running Chasten

Nested Loop Analysis

Check ID	Check Name	File	Matches
PERF001	nested-loops	analyze.py	1
PERF001	nested-loops	display.py	7
PERF001	nested-loops	main.py	0

- **Check ID** → A unique short rule code (e.g., `PERF001`)
- **Check Name** → The rule name that matched (e.g., `nested-loops`)
- **File** → The Python file that the tool scanned (e.g., `analyze.py`)
- **Matches** → Number of times the pattern was detected in that file (e.g., `1` match)

Limitations and Future Directions

- **Limitations of the Chasten Tool**
  - Doesn’t handle style, formatting, or type inference
  - Not optimized for fast use in continuous integration
  - Pattern matches through use of XPath on Python’s AST
- **Empirical Study of Chasten**
  - Frequency of false positives or false negatives?
  - How do students respond to the tool’s feedback?
  - Differences in scores with varied feedback types?

Key Takeaways

- Write declarative rules for AST-based code checks
- Focus on bespoke code structure patterns in Python
- Automated grading aligned with learning outcomes
- Generate data-rich insights into student code patterns

- **Try out Chasten and contribute to its development!**
  - GitHub: https://github.com/AstuteSource/chasten
  - PyPI: https://pypi.org/project/chasten/