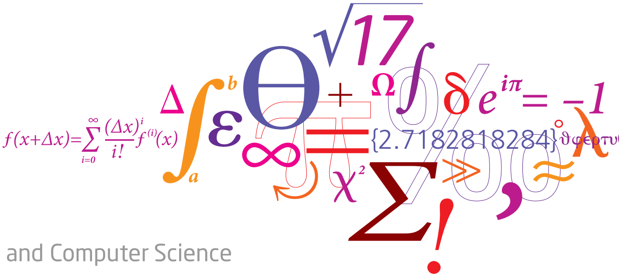
Example slide show

Author



DTU Compute

Department of Applied Mathematics and Computer Science

Observations about online autograders

- They do many things
  - Exercise descriptions
  - Administrative tasks (student onboarding, grading, exports)
  - Calendar-tasks (due-date, handling delays)
  - Assignment handin
  - Plagiarism checking
  - An IDE
  - A CI/CD system (upload code and run)
  - Various forms of automation/ideas about organizing exercises
  - A test system
- I find them hard to use
  - First you parse these 5 numbers from stdin...
  - Blackbox

My goals

- **Best** allow students to find errors and solve their programming problems
- Make the course **software** and **written material** easily maintainable in anticipation of future changes
    - Write things in one place
    - Update concurrently
    - Make sure things don't break
- In this presentation: Inspiration; I am fully aware I did not solve everything.

- ldasf

Best test system for students

- Which test system best allows students to fix problems in their homework?
- Which test system best allow **me** to fix problems in **my code**
- Web-based tests
  - No debugger
  - Local/remote code (upload or edit in bad online IDE)
  - Increased run-time (can tests be run in isolation?)
  - Blackboxing (environment, packages, file locations, how code is called)
    - Drivers
    - Preprocessors
    - Postprocessors
    - Hacks (cat, fix-floats, etc.)
  - Tendency of tests to adopt conventions from the tool (is reading from stdin really something we do?)
- Unittests
  - Debugger
  - Favorite IDE+autocomplete
  - Plugins to all IDEs
  - Student learns relevant skills (unittesting)
  - Test runable in isolation
  - Transparency; everything is python
  - Speed

Casestudy: 02465

- 13 exercise + 3 projects (3 group + individual)
- Fairly involved code (many dependencies)
-

Casestudy: Introduction to python (Vedrana)

- Exam set with 4 problems (Python). Showcase automatic generation of test answers.

Codejudge

- 5 problems
- 11 tests per problem
- 60 files
- 282 lines of code

Unitgrade

- 5 problems
- 11 tests per problem
- 3 files
- n lines of code

Bonus: My version contains a handout stub for students to work with.

Casestudy: Problem set 6

- Problem set 6, create a Fraction class with addition

Codejudge

- 1 problem
- 8 files (.in, .ans)
- 2 tests
- 12 lines of code (input, output; but more code in LaTeX)

Unitgrade

- 1 problems
- 2 files
- 6 tests
- 86 lines of code

Bonus: My version contains more specific tests, generates handout files, contains a solution, and allows automatic checks of implementation for later refactoring.

Casestudy: Exam set 2021

- Exam set for 2021

Codejudge

- 4 problems
- 72 files (!)
- 16 .yml files
- 18 .ans files
- N tests
- 552 lines of code (excluding .ans files)

Unitgrade

- 4 problems
- 6 files (4 problems + 1 test + 1 deploy)
- N tests
- 116 lines of code

Bonus: Automatic checks.

Unittests

- asdf

Developing tests

- Add a report class + deploy script and it works.
- Security and evaluation: Docker + scripts (Download from learn, evaluate/run automatic)
- Support hidden tests

Test development

- No-configuration files approach
- Don't duplicate information

Test development

- No-configuration files approach
- Don't duplicate information

Test development

- No-configuration files approach
- Don't duplicate information

* I think it is important to introduce simplest method first: Test and evaluation. Perhaps use the homework as an example? * you could also use the python course as an example (one problem). * Admin; grading. Setup automatically and evaluate. * Show automatic evaluation directly. print to excel file and .pkl. show autograding. show hidden tests. Show plagiarism checks with moss. Show failed evaluation+log+fix.
Points of the examples: (seperate?) *