LESSON 75

Advanced Debugging and Testing

80 minutes (2 periods) | Software Engineering and Testing | Outcome 1.22 | Outcome 2.19 | Outcome 2.20 | Outcome 2.21 | Outcome 2.22 | Outcome 1.23 | Outcome 3.3

WHAT THIS LESSON DOES

In this lesson, you'll explore advanced debugging and testing techniques for software development. Learn the key stages of testing-unit, integration, and system-and apply them to debug and test a Flask web app with SQLite in VS Code.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
1.22	read, write, test, and modify computer programs
2.19	test solutions and decisions to determine their short-term and long-term outcomes
2.20	identify and fix/debug warnings and errors in computer code and modify as required
2.21	critically reflect on and identify limitations in completed code and suggest possible improvements
2.22	explain the different stages in software testing
1.23	reflect and communicate on the design and development process
3.3	use appropriate programming languages to develop an interactive website that can display information from a database that meets a set of users' needs

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the key stages of software testing and their purposes.
- Develop skills in debugging complex applications using advanced tools.
- Apply unit, integration, and system testing techniques effectively.
- Identify and resolve bugs in a web application environment.
- Evaluate code limitations and propose potential improvements.

BEFORE THE LESSON

- Run through the whole build yourself first: create the venv, pip install flask, drop in app.py, and confirm both planted bugs actually fire (the age type error and the misspelled userss table). A silent success means the starter code drifted.
- Confirm Python and the VS Code Python extension are installed on every machine and that pip install flask succeeds through the school filter. A blocked package index will stall the whole class at step 3.
- Have the full app.py starter code saved somewhere the class can reach, since the lesson only shows a truncated snippet.
- Delete any leftover users.db from your test run so students start clean.

Advanced Debugging and Testing

HOW THE LESSON RUNS

- Introduction
- Understanding Testing Stages
- Create your environment and install Flask
- Set Up the Buggy Flask App
- Identify the Bugs
- Fix the Bugs
- Unit Testing
- Integration Testing
- System Testing
- Wrap Up

TEACHING MOVES

- 2. Understanding Testing Stages. Anchor each stage to the app they are about to build: unit is the insert function alone, integration is the route talking to the database, system is filling the form in the browser and seeing the user listed. Concrete beats definitions.
- 3. Create your environment and install Flask. Explain why the venv exists before they click: it keeps this project's Flask separate from anything else on the machine. Check the .venv folder appears and the terminal prompt shows it is active before anyone runs pip.
- 4. Set Up the Buggy Flask App. Tell them plainly this code contains bugs on purpose and they should not fix anything yet. Get the file saved and the app running so they see it launch before it breaks.
- 5. Identify the Bugs. Demonstrate one breakpoint on the projector: click the gutter, press F5, submit fifteen as an age, and step line by line until it throws. Then let them find the second bug themselves rather than telling them the line.
- 6. Fix the Bugs. Make them explain why int(request.form['age']) fixes the first bug before they type it. The fix is one word; the reasoning is the lesson.
- 7. Unit Testing. Show that a unit test asserts an expected result against an actual one and that a failing assertion is a found bug, not a crash to fear. Have them predict the assertion output before running.
- 8. Integration Testing. Contrast this with the previous step out loud: the unit test called the function directly, the integration test goes through the Flask route. Ask what a route could break that the function alone cannot.
- 9. System Testing. Have them do this by hand in the browser: submit the form, then open View Users and confirm the record appears. Ask why a page that looks right could still be wrong underneath.
- 10. Wrap Up. Ask each student to name one real limitation of the app they just tested, no input validation on the name field for instance, as evidence they can evaluate code rather than only run it.

WHAT TO WATCH FOR

- Students treat form input as the right type, expecting request.form['age'] to be a number when it is always a string. This is the exact first planted bug. At the breakpoint, have them hover the age variable and read the type in the debugger. Seeing 'fifteen' or '15' as a string makes the int() conversion obvious.
- Believing the app is correct because the home page loads and the form submits without an error. A silent wrong answer feels like success. Push them to system test by actually checking the users list after adding, and to write one unit test that asserts against a known value rather than trusting the screen.
- Confusing the three test types, using unit, integration and system interchangeably. Keep the running list on the board: which one calls the function directly, which goes through the route, which uses the browser. Point to the concrete example each time the word comes up.

DIFFERENTIATION

- Emerging: If the venv or pip install is fighting them, pair them with a working machine so they do not miss the debugging, which is the point of the lesson. Give them one bug to focus on with a single breakpoint already placed, and have them narrate what each line does as they step through.
- Developing: They can step through and fix on prompt but not explain the fix. Ask them to write one sentence per bug: what went wrong and why the fix works. Let them predict what the debugger will show before each step press, then check.
- Proficient: Have them add a third bug of their own to the app and write a unit test that catches it, then hand it to a partner to find with the debugger. Push toward Higher Level: ask them to add input validation and a test that proves an invalid age is rejected, the kind of evidence a Coursework Project write-up would use.