LESSON 61

Error Handling and Debugging

80 minutes (2 periods) | Python Web Backend | Outcome 2.20 | Outcome 2.21 | Outcome 2.22 | Outcome 1.22 | Outcome 1.23 | Outcome 3.2 | Outcome 3.3

WHAT THIS LESSON DOES

In this lesson, you'll explore error handling and debugging in Python web development. Learn to manage errors and fix bugs by creating a Flask app, introducing errors, and using try-except blocks and VS Code debugging tools to resolve them.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
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
1.22	read, write, test, and modify computer programs
1.23	reflect and communicate on the design and development process
3.2	create a basic relational database to store and retrieve a variety of forms of data types
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 importance of error handling in building reliable web applications.
- Identify different types of programming errors and their impact on code execution.
- Apply try-except blocks to manage runtime errors effectively.
- Utilise debugging tools in VS Code to trace and resolve issues in code.
- Develop problem-solving skills by simulating and fixing common errors in a Flask app.

BEFORE THE LESSON

- Run the whole sequence yourself first on the school machines: create the venv, pip install flask behind the school filter, run create_db.py, and confirm the app serves at the local address. The filter blocking pip is the most likely thing to stall the room.
- Have the working app.py, create_db.py and users.html ready to hand out so a student whose environment fights them can still reach the debugging steps.
- Check the VS Code Python extension is installed on every machine, since the debugger and Create Environment command both depend on it.

Error Handling and Debugging

HOW THE LESSON RUNS

- Introduction
- Set Up Your Project
- Create Sample Database
- Build Basic Flask App
- What are Errors?
- Introduce a Common Error
- Error Handling with Try-Except
- What is Debugging?
- Debugging in VS Code
- Challenge
- Wrap Up

TEACHING MOVES

- 2. Set Up Your Project. Do this step in lockstep on the projector. The venv creation and Flask install are where the class fragments; get everyone to the same green 'installed' terminal line before moving on.
- 3. Create Sample Database. Have them run create_db.py once and confirm 'Database created successfully!' prints. Warn that running it twice just re-inserts Alice and Bob, which is harmless but confusing if they notice duplicates later.
- 4. Build Basic Flask App. Get a working page showing Alice and Bob before any error is introduced. If it does not render now, nothing later will make sense, so fix it here.
- 5. What are Errors? Anchor each type to something concrete: a missing colon Python catches before running, a wrong database name that crashes mid-run, a wrong answer that runs fine. The distinction between when each is caught matters most.
- 6. Introduce a Common Error. Let them see the raw 500 page and the OperationalError in the terminal on purpose. Point out the terminal, not the browser, tells you what actually went wrong.
- 7. Error Handling with Try-Except. Break the database name again after adding the try-except so they see a controlled message instead of a 500. The contrast with the previous step is the whole point.
- 8. What is Debugging? Contrast print statements with a real debugger before they open one. Frame the debugger as pausing the running program so you can look inside it, not as a magic fix.
- 9. Debugging in VS Code. Flag the reloader must be off for the debugger to attach, and show the wrong_table error happening only when the query runs, reinforcing that this is a runtime error, not a syntax one.
- 10. Challenge. Circulate as they set the breakpoint after fetchall and inspect users. Confirm the app is stopped before they start the debugger, since a still-running server is the usual reason it will not attach.

WHAT TO WATCH FOR

- Students read the browser 500 page and assume it holds the diagnosis, ignoring the terminal traceback. Point them at the terminal every time an error fires. The last line of the traceback names the actual exception; the browser only says something went wrong.
- Students think try-except fixes the error, rather than catching it, so they stop investigating the underlying cause. Show that the wrong database name is still wrong after the except block runs; it just fails politely now. Handling an error and fixing the bug are different jobs.
- Confusing a syntax error with a runtime error, expecting a missing table to be flagged before the app runs. Remind them the wrong_table error only appears when that route is visited and the query executes. Python cannot know the table is missing until it asks the database.

DIFFERENTIATION

- Emerging: If the venv or pip install stalls them, hand over the prepared working project so they can still reach the try-except and debugging steps. Pair them with a student who has a running app and have them narrate what the terminal traceback is saying.
- Developing: Ask them to predict what the terminal will show before they introduce each error, then check against the real output. Have them explain aloud why the reloader must be off before you accept their breakpoint is set.
- Proficient: Ask them to add a second except clause that catches a specific exception type separately, and explain why catching sqlite3.OperationalError is better practice than a bare except. Have them log the error to a file rather than only showing a message, and note this as evidence of reliable design for a coursework write-up.