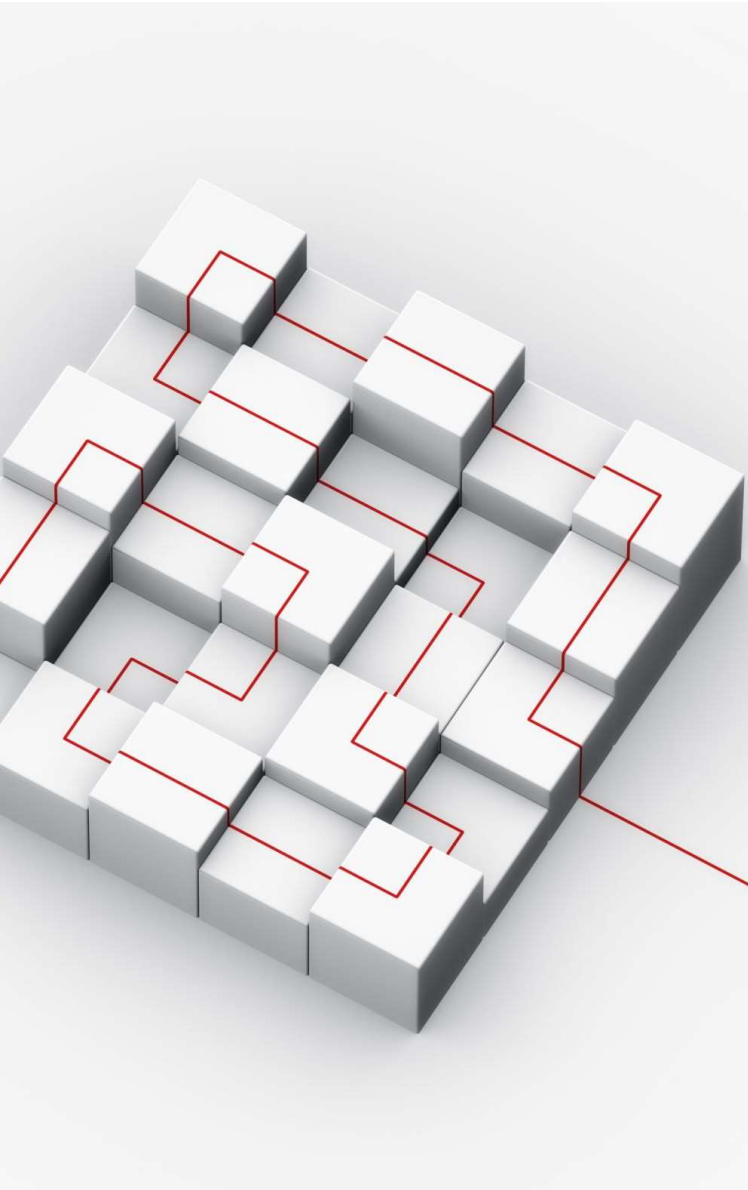


Exceptions and Tools

Designing with Exceptions



This chapter covers

- Nesting Exception Handlers
- Exception Idioms
- Exception Design Tips and Tricks

Nesting Exception Handlers

- Technically, try statements can nest, in terms of both syntax and the runtime control flow through your code.
- When an exception is raised, Python returns to the most recently entered try statement with a matching except clause.
- Because each try statement leaves a marker, Python can jump back to earlier trys by inspecting the stacked markers.

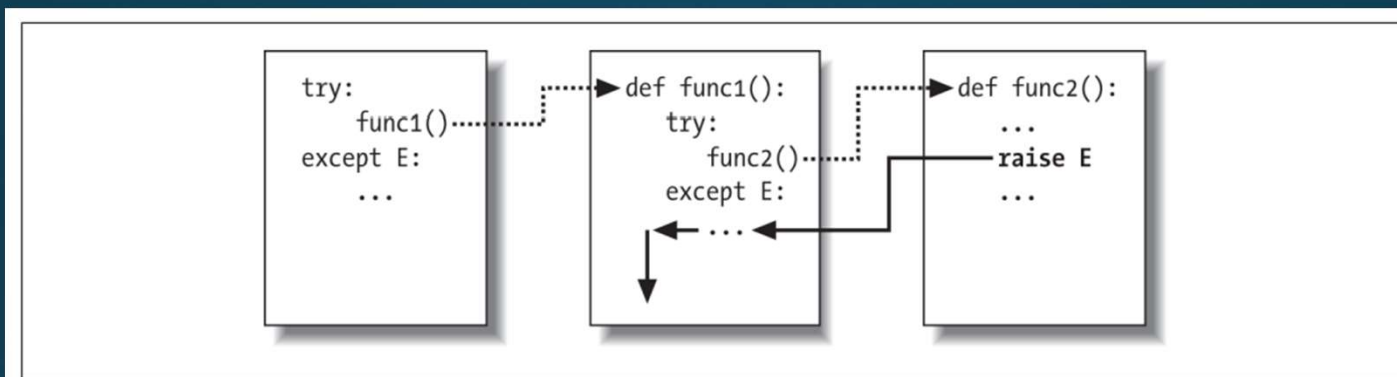


Figure 36-1. Nested try/except statements: when an exception is raised (by you or by Python), control jumps back to the most recently entered try statement with a matching except clause, and the program resumes after that try statement. except clauses intercept and stop the exception—they are where you process and recover from exceptions.

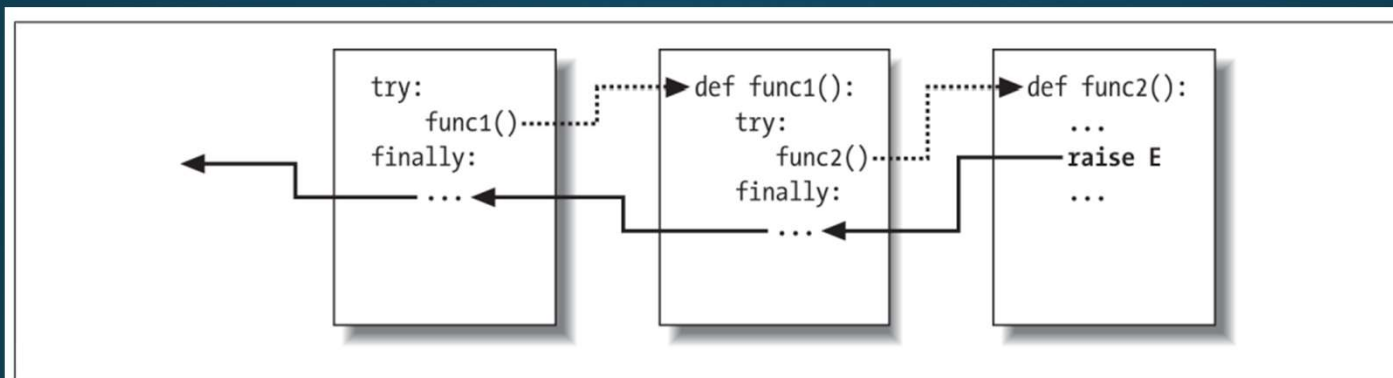


Figure 36-2. Nested try/finally statements: when an exception is raised here, control returns to the most recently entered try to run its finally statement, but then the exception keeps propagating to all finallys in all active try statements and eventually reaches the default top-level handler, where an error message is printed. finally clauses intercept (but do not stop) an exception—they are for actions to be performed “on the way out.”

nestexc.py

Example

Exception Idioms

- Breaking Out of Multiple Nested Loops: “go to”
- Exceptions Aren’t Always Errors
- Functions Can Signal Conditions with raise
- Closing Files and Server Connections
- Debugging with Outer try Statements
- Running In-Process Tests
- More on sys.exc_info
- Displaying Errors and Tracebacks

excidioms.py

Example

Exception Design Tips and Gotchas

- In principle, you could wrap every statement in your script in its own try, but that would just be silly (the try statements would then need to be wrapped in try statements!).
- What to wrap is really a design issue that goes beyond the language itself, and it will become more apparent with use.

Some Recommendations

- Operations that commonly fail should generally be wrapped in try statements.
- However, there are exceptions to the prior rule - in a simple script, you may want failures of such operations to kill your program instead of being caught and ignored.
- You should implement termination actions in try/finally statements to guarantee their execution, unless a context manager is available as a with/as option.
- It is sometimes more convenient to wrap the call to a large function in a single try statement, rather than littering the function itself with many try statements.

Exceptions Gotchas

- Catching Too Much: Avoid Empty except and Exception
 - You've seen that an empty except clause catches every exception that might be raised while the code in the try block runs.
 - That's easy to code, and sometimes desirable, but you may also wind up intercepting an error that's expected by a try handler higher up in the exception nesting structure.
- Catching Too Little: Use Class-Based Categories
 - On the other hand, neither should handlers be too specific.
 - When you list specific exceptions in a try, you catch only what you list.

Page 1162

Exercises

The End