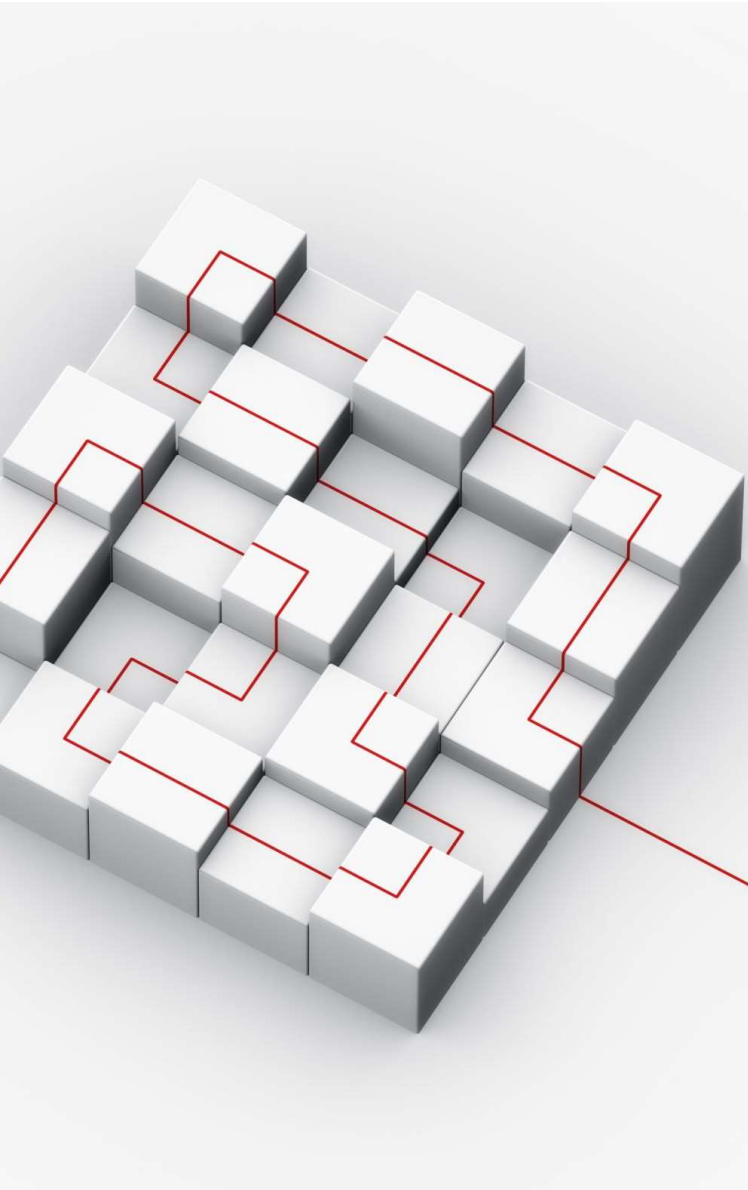


Exceptions and Tools

# Exception Coding Details



# This chapter covers

- The try/except/else Statements
- The try/finally Statements
- Unified try/except/finally
- The raise Statement
- The assert Statement
- with/as Context Managers

# The try/except/else Statement

```
try:
    statements                # Run this main action first
except name1:
    statements                # Run if name1 is raised during try block
except (name2, name3):
    statements                # Run if any of these exceptions occur
except name4 as var:
    statements                # Run if name4 is raised, assign instance raised to var
except:
    statements                # Run for all other exceptions raised
else:
    statements                # Run if no exception was raised during try block
```

# The try/except/else Statement

Clause form	Interpretation
<code>except:</code>	Catch all (or all other) exception types.
<code>except <i>name</i>:</code>	Catch a specific exception only.
<code>except <i>name</i> as <i>value</i>:</code>	Catch the listed exception and assign its instance.
<code>except (<i>name1</i>, <i>name2</i>):</code>	Catch any of the listed exceptions.
<code>except (<i>name1</i>, <i>name2</i>) as <i>value</i>:</code>	Catch any listed exception and assign its instance.
<code>else:</code>	Run if no exceptions are raised in the try block.
<code>finally:</code>	Always perform this block on exit.

# The try/finally Statement

- If a finally clause is included in a try, Python will always run its block of statements “on the way out” of the try statement, whether an exception occurred while the try block was running or not.

```
try:
    statements                # Run this action first
finally:
    statements                # Always run this code on the way out
```

# Unified try/except/finally

```
try:                                # Merged form
    main-action
except Exception1:
    handler1
except Exception2:                 # Catch exceptions
    handler2
...
else:                               # No-exception handler
    else-block
finally:                           # The finally encloses all else
    finally-block
```

```
try -> except -> else -> finally
```

mergedexc.py

# Example

# The raise Statement

- To trigger exceptions explicitly, you can code raise statements.
- Their general form is simple - a raise statement consists of the word raise, optionally followed by the class to be raised or an instance of it.

<code>raise instance</code>	<i># Raise instance of class</i>
<code>raise class</code>	<i># Make and raise instance of class: makes an instance</i>
<code>raise</code>	<i># Reraise the most recent exception</i>



# The assert Statement

- As a somewhat special case for debugging purposes, Python includes the assert statement.
- It is mostly just syntactic shorthand for a common raise usage pattern, and an assert can be thought of as a conditional raise statement.

```
assert test, data          # The data part is optional
```

works like the following code:

```
if __debug__:
    if not test:
        raise AssertionError(data)
```

asserter.py

# Example

# with/as Context Managers

- Python 2.6 and 3.0 introduced a new exception-related statement - the with, and its optional as clause.
- This statement is designed to work with context manager objects, which support a new method-based protocol, similar in spirit to the way that iteration tools work with methods of the iteration protocol.

```
with expression [as variable]:  
    with-block
```

withas.py

# Example

**The End**