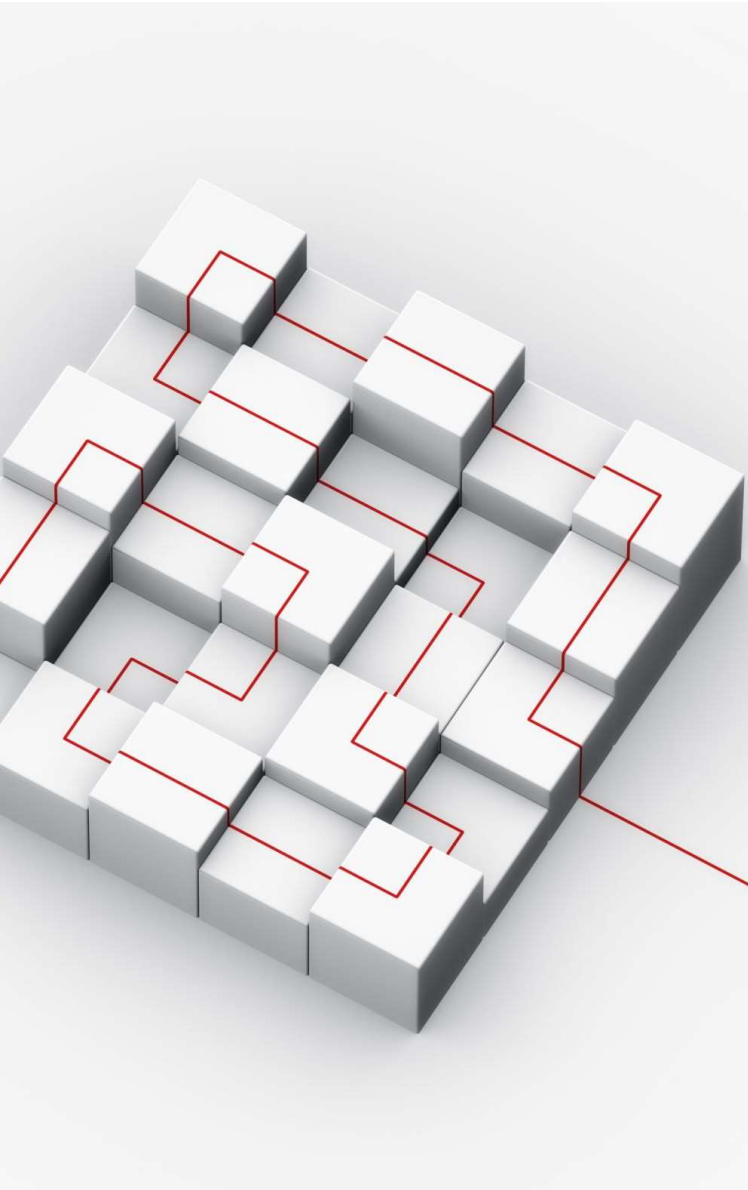


Exceptions and Tools

# Exception Basics



# This chapter covers

- Why Use Exceptions?
- Default Exception Handler
- Catching Exceptions
- Raising Exceptions
- User-Defined Exceptions
- Termination Actions

# Why Use Exceptions?

- In a nutshell, exceptions let us jump out of arbitrarily large chunks of a program.
- One way to think of an exception is as a sort of structured "super go to."
- An exception handler (try statement) leaves a marker and executes some code.

# Exception Roles

- Error handling
- Event notification
- Special-case handling
- Termination actions
- Unusual control flow

# Default Exception Handler

- Because our code does not explicitly catch this exception, it filters back up to the top level of the program and invokes the default exception handler, which simply prints the standard error message.

```
>>> def fetcher(obj, index):  
    return obj[index]
```

```
>>> fetcher(x, 4)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
  File "<stdin>", line 2, in fetcher  
IndexError: string index out of range
```

# Catching Exceptions

- If you don't want the default exception behavior, wrap the call in a try statement to catch exceptions yourself.
- Now, Python jumps to your handler - the block under the except clause that names the exception raised - automatically when an exception is triggered while the try block is running.
- The net effect is to wrap a nested block of code in an error handler that intercepts the block's exceptions.

`basicexceptions.py`

# Example

# Raising Exceptions

- To trigger an exception manually, simply run a raise statement.
- User-triggered exceptions are caught the same way as those Python raises.
- Example: continue with previous demo (basicexceptions.py)



# User-Defined Exceptions

- User-defined exceptions are coded with classes, which inherit from a built-in exception class: usually the class named Exception.

myexceptions.py

# Example

# Termination Actions

- Finally, try statements can say “finally” – that is, they may include finally blocks.
- These look like except handlers for exceptions, but the try/finally combination specifies termination actions that always execute “on the way out,” regardless of whether an exception occurs in the try block or not.
- Example: continue with previous demo ([basicexceptions.py](#))

**The End**