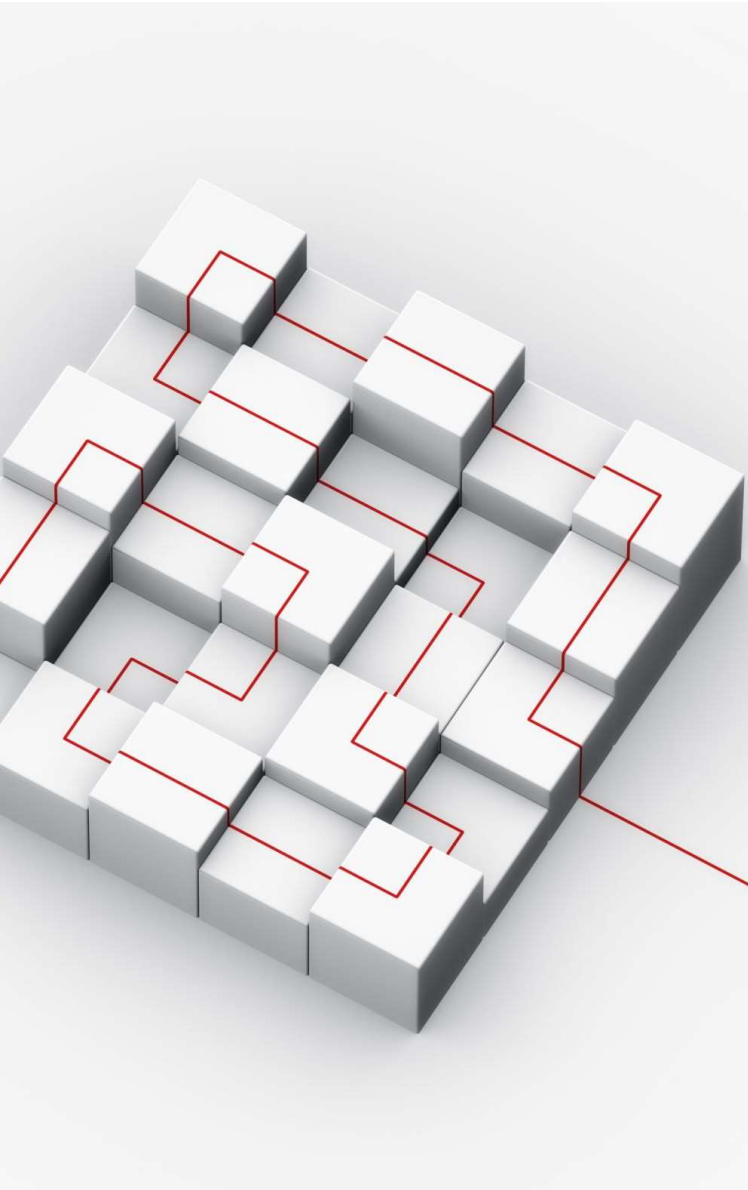


Exceptions and Tools

# Exception Objects



# This chapter covers

- Exceptions
- Why Exception Hierarchies?
- Built-In Exception Classes
- Custom Print Displays
- Custom Data and Behaviour

# Exceptions: Class-Based

- Class-based exceptions better support exception state information (attached to instances) and allow exceptions to participate in inheritance hierarchies (to obtain common behaviors).
- Because they offer all the benefits of classes and OOP in general, they provide a more powerful alternative to the now-defunct string-based exceptions model in exchange for a small amount of additional code.

# String Exceptions (deprecated)

```
C:\code> C:\Python25\python
>>> myexc = "My exception string"
>>> try:
...     raise myexc
... except myexc:
...     print('caught')
...
caught
```

*# Were we ever this young?...*

classexc.py

# Example

# Why Exception Hierarchies?

- For large or high exception hierarchies, however, it may be easier to catch categories using class-based categories than to list every member of a category in a single except clause.
- Perhaps more importantly, you can extend exception hierarchies as software needs evolve by adding new subclasses without breaking existing code.

# Built-in Exceptions

- In Python, all exceptions must be instances of a class that derives from `BaseException`.
- In a try statement with an except clause that mentions a particular class, that clause also handles any exception classes derived from that class (but not exception classes from which it is derived).
- Two exception classes that are not related via subclassing are never equivalent, even if they have the same name.

# Exception Context

- When raising a new exception while another exception is already being handled, the new exception's `__context__` attribute is automatically set to the handled exception.
- An exception may be handled when an `except` or `finally` clause, or a `with` statement, is used.



# Built-in Exception Classes

- `BaseException`: topmost root, printing and constructor defaults
- `Exception`: root of user-defined exceptions
- `ArithmeticError`: root of numeric errors
- `LookupError`: root of indexing error

# Custom Print Displays

- By default, instances of class-based exceptions display whatever you passed to the class constructor when they are caught and printed.
- To provide a more custom display, though, you can define one of two string-representation overloading methods in your class (`__repr__` or `__str__`) to return the string you want to display for your exception.

```
>>> class MyBad(Exception): pass
...
>>> try:
...     raise MyBad('Sorry--my mistake!')
... except MyBad as X:
...     print(X)
...
Sorry--my mistake!
```

customprint.py

# Example

# Custom Data and Behavior

- Built-in exception superclasses provide a default constructor that automatically saves constructor arguments in an instance tuple attribute named args.
- Although the default constructor is adequate for many cases, for more custom needs we can provide a constructor of our own.
- In addition, classes may define methods for use in handlers that provide precoded exception processing logic.

# Why Customized?

- Providing Exception Details
- Providing Exception Methods

custombehavior.py and excparse.py

# Example

**The End**