Exceptions

This is Chapter 8 section 8.2.

The exception construct: a mechanism in programminglanguages for jumping from one point of execution tocode associated with an earlier program block (the handler). A value may bepassed to the handler.

Terminology: exceptions are *raised* or *thrown*; *handled* or*caught*.

Purpose: exceptions allow a function implementor to do a couple ofthings:

- Signal to the caller that the usual kind of result can't be produced
- Shortcut a computation (see example in book)

Exception mechanisms in different languages vary in detail such as what is thrown:

- Specific exception values in ML
- Values in C++
- Strings in Python
- Objects derived from a special class in Python and Java

But you always find

- some way to associate a handler with a block of code. The handler receives control for any matching exception thrown during the lifetime of the associated block (if it is not handled at a lower level)
- some way to raise an exception

Exceptions are handled by the (dynamically) most recentmatching handler. Exceptions handlers are a form of dynamic scoping even instatically scoped languages. Follow the dynamic links in the stack to findmost recent handler (or try-finally).

Some languages offer a finally clause in which youput code to be executed after a block completes, whether it completes normally,a result of return, continue or break, or by raising an an exception. The exceptionneed not be handled in the block associated with the finally clause: it could just be "passing by" on its way to a higher exception handler.

Python exceptions

```
try:   suite 1[except [exception [, value]]:   suite 2]+[else:   suite 3]
```

is a tryblock with multiple handlers. (Note the square bracketshere are meta-characters used to indicate repeating and optional syntacticelements. They are not actually included in Python programs.)

When an exception is raised in suite 1, the first of the suite 2s thatmatch the exception is executed. If none of the handlers match the system continues looking in this blocks caller for a match.

If no exception occurs in suite 1, suite 3 is executed when suite 1 completes.

Try-finally is a way to make sure that some code is executed regardless of whether or not an exception is raised in another block of code.

```
try:   suite 1finally:   suite 2
```

If an unhandled exception occurs in suite 1, suite 2 is executed and then the exceptionis re-raised for handling by higher-level code. There are lots of details such as: what happens if suite 2 itself doesa return or break? Exception is lost

Try-finally solves the problem of how to deallocate resources owned by a frame that is thrown away because of a lower-level exception. Modern languagesalready have some automatic help for this problem:

- C++: destructors are called
- Java, ML, Python rely on GC to release memory resources

but for other resources, e.g. locks and files, some direct instruction fromthe programmer is often necessary. finally clauses give you a place to put codeto solve these problems -- but you have to populate it with the correct code.