

Exceptions Chapter 8 - sec 8.2

Effect: a construct to jump ~~out of~~ from a point of execution to ~~some other~~ code associated with an earlier program block (~~is~~ the handler).
A value may be passed to the handler.

Terminology: exceptions are raised or thrown
handled or caught.

Purpose: allows a function implementor to do a

couple of things:

- o Signal to the caller that ~~a~~ ~~error~~ the usual kind of result can't be produced.
- o Short cut a computation (see example in book).

Exception mechanisms vary in details ~~such as~~ such as "what is thrown" $\left\{ \begin{array}{l} \text{specific exception values in ML} \\ \text{any values in C++} \\ \text{objects in Java derived from a special class in Python and Java} \end{array} \right.$

but you always find

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

Exceptions are handled by the (dynamically) most recent matching handler.

Some languages offer a finally clause in which you put code to be executed after a block completes, whether it completes normally ^{as a result of return, continue, or break} ~~or by handling an exception~~ or by passing on an exception.

Python

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

try block w/ handlers

```
[else:  
    suite 3]
```

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

~~finally~~ when an exception is raised in suite 1, suite 2 is executed then the exception is re-raised ^{itself}

Lots of details - like: what happens if suite 2 does a return or break? exception is lost.

Exceptions ~~and~~ ^{are} handlers ~~provide~~ a form of dynamic scoping even in stat. scoping languages. Follow the dyn. link in the stack to find most recent handler (or finally).

~~Call~~ stack is unwound when an exception is ~~is~~ handled. Stack frames are released. What about resources held in those frames.

C++ destructors are called

Java, ML, Python rely on GC to release memory resources

- What about other resources - esp. locks, files, etc.

- finally clauses can help but you are resp. to populate tho. - correctly.

~~Appt from 8.1~~

Exercises 8.3 & 8.4 are not testable but they are quite interesting

Object-oriented Python - begin looking at Ch. 10 (we will come back to Ch 9)
we will talk about general O-O concepts beginning next time.

```
class <classname> [( <supername [ , supername ]+ )]:  
    suite
```

in the suite put method definitions
always self

```
def m1 (self, p1, p2):  
    }
```

```
def m2 (self, p1 ... p2):  
    }
```

```
def __init__ (self, p1, p2, ..., pn):  
    }
```

Methods' first parameter is ^{in defs.} self
method names must be unique
(no ad-hoc polymorphism).

`--init--` is called immediately after an object is constructed
~~passing~~ ^{explicit} with args passed to the constructor

```
i) class C (par):
```

```
    def __init__ (p1):  
        par.__init__ (self)  
        self.a1 = 1  
        self.a2 = "x"  
        self.a3 = p1
```

```
    def updateupdate (self, p1):  
        self.a3 = p1
```

to create an object
instance = C (42)
instance.a1 == 1 true
instance.update ("42")

Notice: data members ^{of instances} are created in `--init--` method or subseq to creation
Class data members are part of the class suite and act like static members in C++.