What is an exception?

- What happens when procedure execution hits an unexpected condition?
 - Trying to access beyond the limits of a list will raise an IndexError
 - Test = [1,2,3]
 - Test[4]
 - Trying to convert an inappropriate type will raise a TypeError
 - Int(Test)
 - Referencing a non-existing variable will raise a NameError
 - a
 - Mixing data types without appropriate coercion will raise a TypeError
 - 'a'/4
- These are exceptions –exceptions to what was expected

What to do with exceptions?

- What to do when procedure execution is stymied by an error condition?
 - Fail silently: substitute default values, continue
 - Bad idea! User gets no indication results may be suspect
 - Return an "error" value
 - What value to chose? None?
 - Callers must include code to check for this special value and deal with consequences → cascade of error values up the call tree
 - Stop execution, signal error condition
 - In Python: raise an exception

```
raise Exception("descriptive string")
```

Dealing with exceptions

Python code can provide handlers for exceptions

```
try:
    f = open('grades.txt')
    # ... code to read and process grades
except:
    raise Exception("Can't open grades file")
```

 Exceptions raised by statements in body of try are handled by the except statement and execution continues with the body of the except statement

Handling specific exceptions

 Usually the handler is only meant to deal with a particular type of exception. Sometimes we need to clean up before continuing.

Types of exceptions

- Already seen common error types:
 - SyntaxError: Python can't parse program
 - NameError: local or global name not found
 - AttributeError: attribute reference fails
 - TypeError: operand doesn't have correct type
 - ValueError: operand type okay, but value is illegal
 - IOError: IO system reports malfunction (e.g. file not found)

Other extensions to try

• else:

 Body of this clause is executed when execution of associated try body completes with no exceptions

finally:

- Body of this clause is always executed after try,
 else and except clauses, even if they raised
 another error or executed a break, continue or
 return
- Useful for clean-up code that should be run no matter what else happened (e.g. close file)

An example

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError, e:
        print "division by zero! " + str(e)
    else:
        print "result is", result
    finally:
        print "executing finally clause"
```

An example, revised

```
def divideNew(x, y):
    try:
        result = x / y
    except ZeroDivisionError, e:
        print "division by zero! " + str(e)
    except TypeError:
        divideNew(int(x), int(y))
    else:
        print "result is", result
    finally:
        print "executing finally clause"
```