Exceptions

- What to do when procedure execution is stymied by an error condition?
  - Fail silently: substitute default values, continue execution
    - 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.
- See Section 8 of *The Python Tutorial* at docs.python.org

Handling Specific Exceptions

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

```
try:
    f = open(‘grades.txt’)
    # …code to read and process grades
except IOError,e:
    print “Can’t open grades file: ” + str(e)
    sys.exit(0)
except ArithmeticError,e:
    raise ValueError(“Oops, bug in grade calculation! ”
                                 + str(e))
```

Types of Exceptions

- We’ve seen the common errors:

  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 (eg, file not found)

- See Section 6 of *The Python Standard Library* at docs.python.org

Other extensions to `try`

- `else:`
  - Body of this clause is executed when execution of the associated `try` body completes with no exceptions
- `finally:`
  - Body of this clause is always executed after `try`, `else` and `except` clauses, even they've raised another error or executed a `break`, `continue` or `return`.
  - Useful for clean-up code that should be run (e.g., closing files) no matter what else happened.