CSc 120
Introduction to Computer Programming II

Adapted from slides by Dr. Saumya Debray

07: Exceptions
Errors and exceptions in Python

A Python program can have two kinds of errors:

Syntax errors:
• the code is not legal Python syntax
• detected before the program is run

Exceptions:
• the code is legal Python syntax
• but something goes wrong when the program is run

An exception is an error that is only detected at run time.

* This does not count logic errors, which the Python system cannot detect
Some common exceptions

• IOError
  – error in an i/o operation. E.g., a file cannot be opened or disk is full

• IndexError
  – an index into a string or list is out of bounds

• KeyError
  – a non-existent key used to access a dictionary

• TypeError
  – arguments to an operation are of the wrong type

• ValueError
  – type is OK but the value is not. E.g.: int("abc")
Handling exceptions

try
  exception may occur
raise
  exception occurs
catch
  catch and handle the exception
Handling exceptions

Example:

```python
try:
    infile = open(filename)
except:
    print("could not open file: " + filename)
```
Handling exceptions

Example:

```
try:
    code that might raise an exception
except:
    code to handle the exception
```
Handling exceptions

Example:

try:
  code that might raise an exception
except:
  code to handle the exception

• This will catch any exception raised in the try block
• This may not always be desirable
Handling exceptions

```python
>>> def foo(filename):
...     try:
...         infile = open(filename)
...         n = int(infile.read())
...         print("n = " + str(n))
...         print("reciprocal = " + str(1/n))
...     except:
...         print("ERROR: could not read file: " + filename)
```
Handling exceptions

```python
>>> def foo(filename):
...     try:
...         infile = open(filename)
...         n = int(infile.read())
...         print("n = " + str(n))
...         print("reciprocal = " + str(1/n))
...     except:
...         print("ERROR: could not read file: " + filename)
...     ...
```
Handling exceptions

```python
>>> def foo(filename):
...     try:
...         infile = open(filename)
...         n = int(infile.read())
...         print("n = " + str(n))
...         print("reciprocal = " + str(1/n))
...     except:
...         print("ERROR: could not read file: " + filename)
...     ...

>>> foo('file_3')
n = 3
reciprocal = 0.33333333333333333

>>> foo('nonexistent_file')
ERROR: could not read file: nonexistent_file
```
Handling exceptions

```python
>>> def foo(filename):
...     try:
...         infile = open(filename)
...         n = int(infile.read())
...         print("n = " + str(n))
...         print("reciprocal = " + str(1/n))
...     except:
...         print("ERROR: could not read file: " + filename)
...
>>> foo('file_3')
n = 3
reciprocal = 0.3333333333333333

>>> foo('nonexistent_file')
ERROR: could not read file: nonexistent_file

>>> foo('file_0')
n = 0
ERROR: could not read file: file_0
```

The file was read!
The error message doesn't make sense

**CULPRIT:** Catching all exceptions (BAD STYLE)
Handling exceptions

```python
>>> def reciprocal(filename):
    try:
        infile = open(filename)
        n = int(infile.read())
        print("n = " + str(n))
        print("1/n = " + str(1/n))
    except IOError:
        print("ERROR: could not read file: " + filename)
    ...

>>> reciprocal('file_3')
n = 3
1/n = 0.3333333333333333
>>> reciprocal('nonexistent')
ERROR: could not read file: nonexistent
>>> >>> reciprocal('file_0')
n = 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 6, in reciprocal
ZeroDivisionError: division by zero
```
Handling multiple exceptions 1

```python
>>> def reciprocal(filename):
    try:
        infile = open(filename)
        n = int(infile.read())
        print("n = " + str(n))
        print("1/n = " + str(1/n))
    except (IOError, ArithmeticError):
        print("Something broke! :-(")

>>> reciprocal("file_3")
n = 3
1/n = 0.3333333333333333

>>> reciprocal("nonexistent_file")
Something broke! :-(

>>> reciprocal("file_0")
n = 0
Something broke! :-(
```

Handle multiple exceptions in the same way

Behavior for both exceptions is the same
Handling multiple exceptions 2

```python
>>> def reciprocal(filename):
    try:
        infile = open(filename)
        n = int(infile.read())
        print("n = " + str(n))
        print("1/n = " + str(1/n))
    except IOError:
        print("ERROR: could not read file: " + filename)
    except ZeroDivisionError:
        print("ERROR: divide by zero :-(")

>>> reciprocal("file_3")
n = 3
1/n = 0.33333333333333333

>>> reciprocal("nonexistent_file")
ERROR: could not read file: nonexistent_file

>>> reciprocal("file_0")
n = 0
ERROR: divide by zero :-(
```

Handle multiple exceptions in different ways
Handling multiple exceptions 2

```python
>>> def reciprocal(filename):
    try:
        infile = open(filename)
        n = int(infile.read())
        print("n = " + str(n))
        print("1/n = " + str(1/n))
    except IOError:
        print("ERROR: could not read file: " + filename)
    except ZeroDivisionError:
        print("ERROR: divide by zero :-(")

>>> reciprocal("file_3")
n = 3
1/n = 0.3333333333333333

>>> reciprocal("nonexistent_file")
ERROR: could not read file: nonexistent_file

>>> reciprocal("file_0")
n = 0
ERROR: divide by zero :-(
```
Exception propagation

```python
>>> def fun1(x):
    return 1/x

>>> def fun2(x):
    return 1 + fun1(x)

>>> def fun3(x):
    try:
        return 2 * fun2(x)
    except ZeroDivisionError:
        print("caught divide-by-0 in fun3")

>>> fun3(2)
3.0
>>> fun3(0)
catched divide-by-0 in fun3
```
Dealing with exceptions

• If possible and appropriate, try to recover from the exception
  – depends on the problem spec, nature of the exception

• If recovery is not possible, exit the program

```python
import sys

...  
sys.exit(1)  
```

exits the program with error code 1 (this indicates that an error occurred to any other program that may be using this program)
Example

```python
import sys

def read_input(filename):
    try:
        fileobj = open(filename)
    except IOError:
        print("ERROR: could not open file " + filename)
        sys.exit(1)
    for line in fileobj:
        ...process contents of file...
```
Else clause (optional)

Executed if no exceptions are raised.

...  
for fname in names_list:
    try:
        f = open(fname)
    except IOError:
        print("cannot open ", fname)
    else:
        print("length of", fname, "is", len(f.readlines()))
    f.close()
Exceptions: summary

• Avoid naked `except` if at all possible
  – catch and handle specific exceptions by name
  – other exceptions will propagate up to the caller

• Keep the `try ... except` separation as small as possible
  – makes the code easier to understand
  – avoids inadvertent masking of exceptions

• Recover from the exception if possible; otherwise exit with error code 1