Python exceptions

Reuven M. Lerner, PhD
reuven@lerner.co.il

Exceptions

```
a = []

a[5]

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

IndexError: list index out of range
```

- What is IndexError?
  - An exception

Exceptions

- Cannot be mixed up with a returned value, such as 0 or -1
- Lots of specific types of exceptions
- You can trap them, and then decide what to do
- You can create your own

Trapping exceptions

```
a = [1,2,3]

try:

  a[5]

except:

  print "You triggered an exception"
```

Trapping exceptions

- Put potentially dangerous/problematic code inside of a “try” block
- After the code runs, use an “except” block
- That traps all exceptions
- You can then decide what to do — print an error message, raise a new exception, log to a file, or nothing at all

raise

```
raise IndexError

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

IndexError
```

```
raise IndexError("Actually, you're fine.")

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

IndexError: Actually, you're fine.
```

Trapping specific exceptions

```
try:
  raise IndexError
except ZeroDivisionError:
  print "Divided by zero"
except (IndexError, EOFError):
  print "Index or format error"
except:
  print "Other error"
```

Get the exception object

```
try:
  raise IndexError
except ZeroDivisionError as e:
  print "Divided by zero"
except (IndexError, EOFError) as e:
  print "Index or format error"
except Exception as e:
  print "Other error"
```

else

- You can add an “else” clause to your exception handling!
- This code executes if no exception was fired

finally

- You can also include a “finally” clause in your exception handling
- If that clause exists, then it will always execute, regardless of whether the exception was raised

finally example

```
filename = raw_input("Enter a filename: ")

try:
  f = open(filename)
  print len(f.read())
except:
  print "Cannot open the file"
finally:
  print "Whew, glad that's over!"
```

```
try:
    print sys
except IOError as e:
    print "IOError,{}".format(e)
except ZeroDivisionError as e:
    print "ZDV: {}".format(e)
except:
    print "Other exception"
else:
    print "Nothing bad happened; in else"
finally:
    print "In finally"
```

"finally" always runs!

```
def foo():
  try:
      print "first"
      return 5
  except:
      print "There was an exception"
  finally:
      print "I'm in finally"
```

Built-in exception classes

- http://docs.python.org/library/exceptions.html
- Classes and subclasses — so you can trap for a parent class, or a more specific subclass
- ArithmeticError
- IOError
- RuntimeError
- SyntaxError

Custom exception classes

```
class ReuvenException(Exception):

   pass

raise ReuvenException

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

__main__.ReuvenException
```

Exception info

```
try:
    10/0
except Exception as e:
    print e.args     # tuple
    print e.message  # string
```

Traceback

- What if you want the traceback that Python programs display?
- Use the traceback module, which provides an interface to the most recent error and traceback
- Python 3.x finally fixed this problem; the exception object (instance) itself now includes traceback info

Print traceback

```
try:
    print 5/0
except:
    traceback.print_exc()
```

Using alternatives

- Sometimes, Python developers might need to consider using a number of alternatives
- For example, maybe you want to use cStringIO or StringIO
- Use exceptions!

```
try:
    from cStringIO import StringIO
except ImportError:
    from StringIO import StringIO
```

Exception cost

- In many languages, raising (or catching) an exception is an expensive operation
- Not so in Python — feel free to use exceptions whenever they're appropriate