

Day 4: I/O and Exceptions

Suggested reading: *Learning Python* (3rd Ed.)

Chapter 9: Tuples, Files, and Everything Else

Chapter 27: Exception Basics

Turn In Homework

Homework Review

**Write code.
At least a little.
Every day.
Play around!**

Files

Preparing to Read a File

```
my_file = open(filename, 'r')
```

- *filename* can be absolute or relative
- 'r' means "read", is the default, can be omitted

```
data = open('seq_03_T.txt', 'r')
```

```
parameters = open(parameter_filename)
```

- when done with file, close it

```
my_file.close()
```

Reading One Line at a Time

```
for line in file_object:  
    # Note: most lines have newline at end  
    print line.rstrip('\n')
```

```
total = 0  
count = 0  
input = open('my-data.txt')  
for line in input:  
    total += int(line)  
    count += 1  
input.close()  
mean = float(total) / float(count)  
print 'Mean value = %.1f' % (mean)
```

Reading Whole Files

```
line_list = file_object.readlines()
```

- One list element per line
- Trailing newlines on each element

```
input = open('name-list.txt')  
lines = input.readlines()  
input.close()  
  
names = set()  
for line in lines:  
    set.add(line.rstrip('\n'))  
print '%d names, %d unique' % \  
    (len(lines), len(names))
```


Digression #1: Failure

Run Time Failures

- Sometimes, things go badly at run-time
- Have seen error messages like this already:

```
>>> a[2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

- Working with files is particularly prone to failures

```
f = open('this_filename_does_not_exist')
```

- Missing file
- Not allowed to read from or write to file
- Disk full when writing file

Exceptions

- *Raised* when a run-time failure occurs
- Allows Python to get out of arbitrarily deep code
- *YOU* decide when and where to *handle* exceptions
- Can raise your own
- *Exceptions are objects!!!*

```
BaseException
Exception
    StandardError
        ArithmeticError
            FloatingPointError
            OverflowError
            ZeroDivisionError
        AssertionError
        AttributeError
        BufferError
        EOFError
        EnvironmentError
            IOError
            OSError
```

```
...
```

Handling Exceptions

```
try:
    # Living dangerously
except ExceptionTypeA, e:
    print 'Caught exception A:', e
except ExceptionTypeB, e:
    print 'Caught exception B:', e
print 'Continue here'
```

- If code in **try** block raises exception:
 - Check **except** clauses in order
 - Exception variable (e.g., **e**) may contain extra info
- Execution continues after last **except** block

Not Handling Exceptions



```
bar()
```



```
def bar:  
    foo()
```



```
def foo:  
    raise ValueError()
```

Back to Files

Reading a File Carefully

```
try:
    input = open(filename)
    data = input.readlines()    # or other...
    input.close()
except IOError, e:
    print 'Cannot read', filename
    print e.strerror    # see help(IOError)

# What happens here if the read failed?
```

Writing a File

- Prepare to create (or overwrite):

```
output = open(filename, 'w')
```

- Prepare to append (or create):

```
output = open(filename, 'a')
```

- Options for writing data to a file:

```
output.write('Must add newline!\n')
```

```
output.writelines(list_of_strings)
```

```
print >>output, 'No newline here'
```


Writing a File Carefully

```
try:
    output = open(filename, 'w')
    output.writelines(data_list)      # e.g.
    output.close()
except IOError, e:
    print 'Cannot write', filename
    print 'Error:', e      # different output

# What happens here if the write failed?
```

Digression #2: Modules

Brief Introduction to Modules

```
import sys
# help(sys)
print 'Python', sys.version
```

- Extra functionality is bundled into *modules*
- To use functionality, must **import** the module
- Prefix functions or data with *module name + dot*

Standard File Objects

- `sys.stdin` = standard (interactive) input
- `sys.stdout` = standard output (default for `print`)
- `sys.stderr` = standard error (alternative output)

```
import sys

sys.stdout.write('Enter name: ') # no \n
input = sys.stdin.readline().strip()
if input != 'Tim Cartwright':
    print >>sys.stderr, 'Wrong person!'
    sys.exit(1)
```

Directories

File Test Operators

path exists	<code>os.path.exists(<i>path</i>)</code>
path is absolute (starts with /)	<code>os.path.isabs(<i>path</i>)</code>
path is regular file	<code>os.path.isfile(<i>path</i>)</code>
path is directory	<code>os.path.isdir(<i>path</i>)</code>
path is symbolic link	<code>os.path.islink(<i>path</i>)</code>
join path parts	<code>os.path.join(<i>p1</i>, <i>p2</i>, ...)</code>
directory part of path	<code>os.path.dirname(<i>path</i>)</code>
filename part of path	<code>os.path.basename(<i>path</i>)</code>

```
import os.path
if os.path.exists(filename):
    if os.path.isdir(filename):
        print 'Skipping dir', filename
    else:
```

Directory Contents I

```
import os
files = os.listdir(directory)
```

- All entries in directory, except `.` or `..`
- Arbitrary order

```
import os
import os.path

for e in sorted(os.listdir('.')):
    if os.path.isdir(e): print 'dir:', e
    elif os.path.isfile(e): print 'file:', e
    else: print 'other:', e
```

Shell-Like Operations

create a directory	<code>os.mkdir(<i>path</i>, <i>mode</i>)</code>
create a directory recursively	<code>os.makedirs(<i>path</i>, <i>mode</i>)</code>
remove file	<code>os.remove(<i>path</i>)</code>
rename/move a path	<code>os.rename(<i>old</i>, <i>new</i>)</code>
remove (empty) directory	<code>os.rmdir(<i>path</i>)</code>
change permissions	<code>os.chmod(<i>path</i>, <i>mode</i>)</code>
change ownership	<code>os.chown(<i>path</i>, <i>uid</i>, <i>gid</i>)</code>
create a symlink	<code>os.symlink(<i>path</i>, <i>link</i>)</code>
read the path from a symlink	<code>os.readlink(<i>path</i>)</code>

Last 2 Slides!

Other Scripting Languages

- Most have similar I/O operations
- Check for different or additional:
 - Operation names (**-d** vs. **isdir()** vs. **directory?()**)
 - Syntax
 - Operations
- Not all languages have (real) exceptions (e.g., Perl)

Homework

- Read a large file of words
- Do a word frequency count and report
 - What collection type works best here?
 - Consider subsetting the full dataset for testing
- BE SURE TO LABEL YOUR PRINTOUT!!!

```
#!/usr/bin/env python
```

```
"""Homework for CS 368-4 (2011 Fall)
Assigned on Day 04, 2011-11-03
Written by <Your Name>
"""
```