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

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

- `filename` can be absolute or relative
- `'r'` means “read”, is the default, can be omitted

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

- when done with file, close it

```python
my_file.close()
```
Reading One Line at a Time

```python
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

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

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

```python
input = open('name-list.txt')
lines = input.readlines()
input.close()

names = set()
for line in lines:
    set.add(line.rstrip('
'))
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:

```python
>>> 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

```python
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
  ...

```python
Exception
```
Handling Exceptions

```python
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

```python
def foo:
    raise ValueError()

def bar:
    foo()

bar()
```
Back to Files
Reading a File Carefully

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

- Prepare to append (or create):
  
  ```python
  output = open('filename', 'a')
  ```

- Options for writing data to a file:
  
  ```python
  output.write('Must add newline!\n')
  output.writelines(list_of_strings)
  print >>output, 'No newline here'
  ```
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

```python
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)

```python
import sys

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

- path exists: `os.path.exists(path)`
- path is absolute (starts with `/`): `os.path.isabs(path)`
- path is regular file: `os.path.isfile(path)`
- path is directory: `os.path.isdir(path)`
- path is symbolic link: `os.path.islink(path)`
- join path parts: `os.path.join(p1, p2, …)`
- directory part of path: `os.path.dirname(path)`
- filename part of path: `os.path.basename(path)`

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

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

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

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