INFO1110 & COMP9001: Introduction to Programming

School of Information Technologies, University of Sydney

Copyright Warning

Lecture 12: File Input/Output

Open, Read/Write, Close

Files

Files are an idea that makes information storage simple for users.

What kinds of file are these?

- hello.py
- family.jpg
- addresses.db
- birthdaylist.txt

Files contain information

There are no rules about what information is stored in a file. Text, images, binary data

The file name suffix[1] is there for the operating system to *identify* which program should be associated when opening the file.

A unix/linux tool called `file` can scan the contents of a file and determine its type

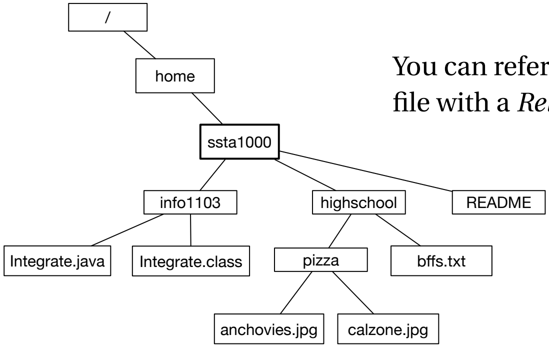
```
~> file HelloWorld.java
HelloWorld.java: ASCII C++ program text
~> file HelloWorld.class
HelloWorld.class: compiled Java class data, version 50.0 (Java 1.6)
~> file runButton.png
runButton.png: PNG image, 30 x 24, 8-bit/color RGBA, non-interlaced
~> file hello.py
hello.py: ASCII text
~> file hello.pyc
hello.cpython-36.pyc: data
```

[1] those final letters after the full stop

Location of Files

There is a *path* associated with files

You can refer to the
file with a *Relative* path, or *Absolute* path



Where is calzone.jpg?

File I/O

To read from or write to from a file you need several things

1. The file has to be there
2. The file has to be available — it must be opened
3. You must have access to it
4. You must know what to read/write

Once you've finished with a file you should *always* close it.

Creating a `File` Object

First we need to create an File Object

`File` object: *An abstract representation of a file.*

Consider the example where we open the file called README

```
file_variable = open("README", "r")
```

What can go wrong here?
Will this compile?
Will this run?

Accessing the `File`

Next we will need some kind of access to it. There are *many* different ways to access a file, but the easiest is the `readline()` method

`readline()`: A method to read exactly one line of text that is delimited by a new line character (default). A String object is returned. Where the string object is empty, there is no more data in the file

```
file_variable = open("README", "r")
one_line = file_variable.readline()
print("first line of file is : " + one_line)
```

Something is missing in this code

Accessing the `File` (cont.)

There are *uncaught exception* that “must be caught” or dealt with when using files.

This is where the idea of *exceptions* are important.

Suppose you have to read one integer from a file called numbers.txt and print it to console

```
12
```

What could go wrong with the following code?

Reading Integer from File

```python
import sys

# read in one line of the file, convert to an integer and print
# includes error correction
the_file = open(sys.argv[1], 'r')
the_integer = int(the_file.readline())
the_file.close()

print("the number is:" + str(the_integer))
print("all done")
```

Compiles? Works?

Reading Integer from File

Array bounds checking

Expecting file to exist

Expecting data to be there

Always expect integer

Didn’t close file

Reading Integer from File

```python
import sys
# read in one line of the file, convert to an integer and print
# includes error correction
try:
    the_file = open(sys.argv[1], 'r')
    the_integer = int(the_file.readline())
    the_file.close()
except IndexError:
    pass
except FileNotFoundError:
    pass
except TypeError:
    pass
except NoMoreDataError:
    pass
finally:
    pass

print("the number is:" + str(the_integer))
print("all done")
```

Compiles? Works?

Writing to File

Switch the flag to open the file in write mode

```python
the_file = open(sys.argv[1], 'w')
```

`f.write(string)` writes the contents of string to the file, returning `None`. Don't forget the new line character!

```python
the_file = open(sys.argv[1], 'w')
the_file.write("10")
the_file.close()
```

Potential data loss - If the file exists then it will be truncated to zero size; otherwise, a new file will be created.

Writing numbers to File

```python
import sys
import math

def write_numbers(outfile, numbers):
    if outfile == None or numbers == None:
        return

    for num in numbers:
        the_string = "{}\n".format(num)
        outfile.write(the_string)

outfilename = "numbers_output.txt"
outfile = open(outfilename, 'w')
numbers = [ 1.0, 3.14, math.sqrt(2), 14.0/1.0 ]
write_numbers(outfile, numbers)
outfile.close()
```

Reading numbers from File

```python
import sys
import math

def read_numbers(infile, numbers):
    if infile == None or numbers == None:
        return

    done = False
    while not done:
        line = infile.readline()
        if not line:
            break
        numbers.append( int(line) )

infilename = "numbers_input.txt"
infile = open(infilename, 'r')
numbers_in = []
read_numbers(infile, numbers_in)
infile.close()
print("numbers in:")
print(numbers_in)
```

Reading Text Files

Often a file contains many different parts. These need to be loaded into memory for the program to do useful work.

Example: read a file and separate the numerical data from text

The following file is "points.txt", it contains 2D point data of exactly 20 locations

```
4, 12
5, 3
18, 19
43, 27
...
140, 0
```

You are to extract the coordinates and store them in a list of 2-tuples

```
[ (4,12), (5,3), (18,19), ..  (140,0) ]
```

Reading Point data

```
def read_points(infilename):
    points = [None]*20
    location = 0
    line_num = 0
    try:
        infile = open(infilename , 'r')

        done = False
        while not done:

            line = infile.readline()
            if not line:
                break
            line_num += 1

            # ???

        infile.close()
    except FileNotFoundError as fnfe:
        myerr("file {} not found\n".format(infilename))
```

Reading Point data

```python
# extract tokens from one line
tokens = line.split(',')
if len(tokens) < 2:
    # bad line, skip to next line
    myerr("less than 2 tokens on line #{}\n".format(line_num))
    continue

# parse integers from 1st and 2nd tokens
try:
    x = int(tokens[0].strip())
    y = int(tokens[1].strip())

    # create a new point with data
    pair = (x,y)
    # update the array
    points[location] = pair;
    location = location + 1;

except ValueError as ve:
    # bad number, skip to next line
    myerr("could not convert to int on line #{}\n".format(line_num))
    continue
```

Reading Point data

What is the output when using the previous text file `points_perfect.txt`:

```
    ~> python PointFileReader.py points_perfect.txt
```

If we change the input file

```
1, 2
3,
4  4
a , 5
         8            ,    7
1600, , 14 ...

9, 14, 32, 57
# frivolous comments!
```

What is the output?

Reading Point data

```python
def myerr(s):
    sys.stderr.write(s)

def read_points(infilename):
    points = [None]*20
    location = 0
    line_num = 0
    try:
        infile = open(infilename , 'r')

        done = False
        while not done:

            line = infile.readline()
            if not line:
                break
            line_num += 1

            # extract tokens from one line
            tokens = line.split(',')
            if len(tokens) < 2:
```

Reading Point data (cont.)

```
                # bad line, skip to next line
                myerr("less than 2 tokens on line #{}\n".format(line_num
                continue

            # parse integers from 1st and 2nd tokens
            try:
                x = int(tokens[0].strip())
                y = int(tokens[1].strip())

                # create a new point with data
                pair = (x,y)

                # update the array
                points[location] = pair;
                location = location + 1;

            except ValueError as ve:
                # bad number, skip to next line
                myerr("could not convert to int on line #{}\n".format(li
                continue

```

Reading Point data (cont.)

```

        infile.close()

        print("read {} lines".format(line_num))
        print("successfully entered {} points".format(location))

        i = 0
        while i < 20:
            if points[i] != None:
                print("point[{index}]: {x} {y}".format(
                        index=i, x=points[i][0], y=points[i][1]))
            i += 1

    except FileNotFoundError as fnfe:
        myerr("file {} not found\n".format(infilename))

#read_points("points_perfect.txt")
#read_points("points_missing.txt")
#read_points("points_non_numbers.txt")
read_points("points_horror.txt")
```