DATA 520
Lecture 14
Reading and Writing Files II

Formatting Data

Reading one file, appending to another file

(will not run)

```python
# totalfilenumbers.py

def sum_number_pairs(input_file, output_filename):
    """ (file open for reading, str) -> NoneType
    Read the data from input_file, which contains two floats per line
    separated by a space. Open file named output_file and, for each line in
    input_file, write a line to the output file that contains the two floats
    from the corresponding line of input_file plus a space and the sum of the
    two floats.
    """

    with open(output_filename, 'w') as output_file:
        for number_pair in input_file:
            number_pair = number_pair.strip()
            operands = number_pair.split()
            #return(str(operands))
            total = float(operands[0]) + float(operands[1])
            new_line = '{0} {1}'.format(number_pair, total)
            output_file.write(new_line)
```

```
# number_pairs.txt
1.3 3.4
2 4.2
-1 1
```

Homework 10

Was due before class Wednesday October 18 (NOW)!

Make it run!

totalfilenumbers.py

Reading one file, writing to another file

```
def sum_number_pairs(readfile, writefile):
    """ (file for reading, file for writing, str) -> NoneType

    Read the data from readfile, which contains two floats per line separated by a space. Open file named writefile and, for
    each line in readfile, write a line to the output file that contains the two floats from the corresponding line of readfile
    plus a space and the sum of the two floats.

    readfile:
    1.3 3.4
    2 4.2
    -1 1
    1.3 3.4
    2 4.2

    writefile after processing (with optional formatting):
    1.3 + 3.4 = 4.7
    2 + 4.2 = 6.2
    -1 + 1 = 0.0
    1.3 + 3.4 = 4.7

    """
```

Reading one file, writing to another file

```
def sum_number_pairs(readfile, writefile):
    #comments stripped

    with open(readfile, 'r') as input_file: # open file for reading
        with open(writefile, 'w') as output_file: # open file for writing
            for number_pair in input_file: # for each line in file
                number_pair = number_pair.strip() # remove newline
                operands = number_pair.split()  # now two numbers in a list
                total = float(operands[0]) + float(operands[1])  # get sum
                # three ways to do it
                #new_line = '{0} {1}'.format(number_pair, total)
                #new_line = str(operands[0]) + '\t+\t' + str(operands[1]) + '\t=\t' + str(total)
                new_line = str(operands[0]) + ' + ' + str(operands[1]) + ' = ' + str(total)
                output_file.write(new_line +'\n')
```

Reading one file, writing to another file

What if we do not want to overwrite a file?

```
def sum_number_pairs(readfile, writefile):
    #comments stripped

    # a simple way based on what we know
    if open(writefile, 'r'):
        return 'The write file is already there!'  # if return, no further code is executed

    # another way using the os (operating system) module
    import os.path
    if os.path.isfile(writefile):
        return 'The write file is already there!'  # if return, no further code is executed

    # how to ask if the user wants to overwrite?
```

Reading one file, writing to another file

What if we do not want to overwrite a file?

```
import os.path
def sum_number_pairs(readfile, writefile):
    #comments stripped

    # how to ask if the user wants to overwrite?
    if os.path.isfile(writefile):
        overwrite = input('The file exists. Do you want to overwrite it? (y/n)')
        if overwrite == 'y':
            print ('I will overwrite the ' + writefile + ' file.') # further code executed below
            # further code executed below

        elif overwrite == 'n':
            return 'The write file is already there and you chose not to overwrite it.' # returns (exits)
        else:
            return 'Invalid key pressed' # returns (exits)

    print('Writing over file') # etc.
```

Reading one file, writing to another file

What if we do not want to overwrite a file? (while)

```
import os.path
def sum_number_pairs(readfile, writefile):
    #comments stripped

    # how to ask if the user wants to overwrite?
    while os.path.isfile(writefile):
        overwrite = input('The file exists. Do you want to overwrite it? (y/n)')
        if overwrite == 'y':
            print ('I will overwrite the ' + writefile + ' file.') # further code executed below
            break # further code executed below
        elif overwrite == 'n':
            return 'The write file is already there and you chose not to overwrite it.' # returns
        else:
            print('Invalid key pressed')
            continue

    print('Writing over file') # etc.
```

Reading a data file

Pseudocode

```
Skip the first line in the file   # you know the data layout, first line is descriptor
Find and process the first line of data in the file: comment?  # begin with '#' ?
For each of the remaining lines:
    Process the data on that line  # read those values
```

first function in of time_series_read.py: (book has time_series.py)

```python
def skip_header(reader):
    """ (file open for reading) -> str
    Skip the header in reader and return the first real piece of data.
    """

    # Read the description line - you know the data!
    line = reader.readline()

    # Find the first non-comment line
    line = reader.readline()
    while line.startswith('#'):
        line = reader.readline()

    # Now line contains the first real piece of data
    return line
```

Reading a data file

Second function in time_series_read.py: (book has time_series.py)

```
def process_file(reader):
    """ (file open for reading) -> NoneType
    Read and print the data from reader, which must start with a single
    description line, then a sequence of lines beginning with '#', then a
    sequence of data.
    """

    # Find and print the first piece of data
    line = skip_header(reader).strip() # calls function above
    print(line)

    # Read the rest of the data
    for line in reader:
        line = line.strip()
        print(line)

if __name__ == '__main__':
    with open('hopedale.txt', 'r') as input_file:
        process_file(input_file)
```

combine into time_series_read.py and run...

Reading a data file

Now we will create another function in another file

```
# read_smallest.py
import time_series_read # so we can use functions inside time_series_read.py
def smallest_value(reader):
    """ (file open for reading) -> NoneType
    Read and process reader and return the smallest value after the
    time_series header.
    """
    line = time_series_read.skip_header(reader).strip()
    # Now line contains the first data value; this is also the smallest value
    # found so far, because it is the only one we have seen.
    smallest = int(line)
    for line in reader:
        value = int(line.strip())
        # If we find a smaller value, remember it.
    if value < smallest:
        smallest = value

    return smallest

if __name__ == '__main__':
    with open('hopedale.txt', 'r') as input_file:
        print(smallest_value(input_file))
```

save as read_smallest.py and run...

Reading a data file

Sometimes a data point is missing, coded as '.', 99, 999, NULL, NA, '-'

- sometimes there are blank lines, typos, stray symbols, etc.

save as hebron.txt:

```
Coloured fox fur production, Hebron, Labrador, 1834-1839
#Source: C. Elton (1942) "Voles, Mice and Lemmings", Oxford Univ. Press
#Table 17, p.265--266
#remark: missing value for 1836
55
262
-
102
178
227
```

in the console:

```
import read_smallest
read_smallest.smallest_value(open('hebron.txt', 'r'))
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    read_smallest.smallest_value(open('hebron.txt', 'r'))
  File "C:\Users\Steve9\AppData\Local\Programs\Python\Python35-32\read_smallest.py", line 13, in smallest_value
    value = int(line.strip())
ValueError: invalid literal for int() with base 10: '-'
```

Reading a data file

So modify the program according to Gries

```python
# read_smallest.py
import time_series_read
def smallest_value(reader):
    """ (file open for reading) -> NoneType
    Read and process reader and return the smallest value after the
    time_series header.
    """
    line = time_series_read.skip_header(reader).strip()
    # Now line contains the first data value; this is also the smallest value
    # found so far, because it is the only one we have seen.
    smallest = int(line)
    for line in reader:
        line = line.strip()
        if line != '-':
            value = int(line.strip())
            # If we find a smaller value, remember it.
            if value < smallest:
                smallest = value

    return smallest

if __name__ == '__main__':
    with open('hopedale.txt', 'r') as input_file:
        print(smallest_value(input_file))
```

save and run…

Reading a data file

The program according to Gries: after we add a missing value indicator at the front of hopedale.txt

```python
# read_smallest.py
import time_series_read
def smallest_value(reader):
    """ (file open for reading) -> NoneType
    Read and process reader and return the smallest value after the
    time_series header.
    """
    line = time_series_read.skip_header(reader).strip()
    # Now line contains the first data value; this is also the smallest value
    # found so far, because it is the only one we have seen.
    smallest = int(line)  # first value always expected to be a number
    for line in reader:
        line = line.strip()
        if line != '-':
            value = int(line.strip())
            # If we find a smaller value, remember it.
            if value < smallest:
                smallest = value

    return smallest

if __name__ == '__main__':
    with open('hopedale.txt', 'r') as input_file:
        print(smallest_value(input_file))
```

Reading a data file

Space-delimited data, multiple lines

lynx.dat:

```
Annual Number of Lynx Trapped, MacKenzie River, 1821-1934
#Original Source: Elton, C. and Nicholson, M. (1942)
#"The ten year cycle in numbers of Canadian lynx",
#J. Animal Ecology, Vol. 11, 215--244.
#This is the famous data set which has been listed before in
#various publications:
#Cambell, M.J. and Walker, A.M. (1977) "A survey of statistical work on
#the MacKenzie River series of annual Canadian lynx trappings for the years
#1821-1934 with a new analysis", J.Roy.Statistical Soc. A 140, 432-436.
  269.  321.  585.  871. 1475. 2821. 3928. 5943. 4950. 2577.  523.   98.
  184.  279.  409. 2285. 2685. 3409. 1824.  409.  151.   45.   68.  213.
  546. 1033. 2129. 2536.  957.  361.  377.  225.  360.  731. 1638. 2725.
 2871. 2119.  684.  299.  236.  245.  552. 1623. 3311. 6721. 4245.  687.
  255.  473.  358.  784. 1594. 1676. 2251. 1426.  756.  299.  201.  229.
  469.  736. 2042. 2811. 4431. 2511.  389.   73.   39.   49.   59.  188.
  377. 1292. 4031. 3495.  587.  105.  153.  387.  758. 1307. 3465. 6991.
 6313. 3794. 1836.  345.  382.  808. 1388. 2713. 3800. 3091. 2985. 3790.
  674.   81.   80.  108.  229.  399. 1132. 2432. 3574. 2935. 1537.  529.
  485.  662. 1000. 1590. 2657. 3396.
```

Reading a data file

Space-delimited data, multiple lines

Pseudocode:

```
Find the first line containing real data after the header
For each piece of data in the current line:
    Process that piece of data


For each of the remaining lines of data:
For each piece of data in the current line:
    Process that piece
```

Reading a data file

Space-delimited data, multiple lines

first part of read_spaced_data.py

```
# read_spaced_data.py
import time_series_read
def find_largest(line):
    """ (str) -> int
    Return the largest value in line, which is a whitespace-delimited string
    of integers that each end with a '.'.
    >>> find_largest('1. 3. 2. 5. 2.')
    5
    """

    # Set the largest value, to be replaced no matter what.
    largest = -1
    for value in line.split():
        # Remove the trailing period.
        v = int(value[0:-1])  # I added a zero, from first to last
        # If we find a larger value, remember it.
        if v > largest:
            largest = v

    return largest
```

Reading a data file

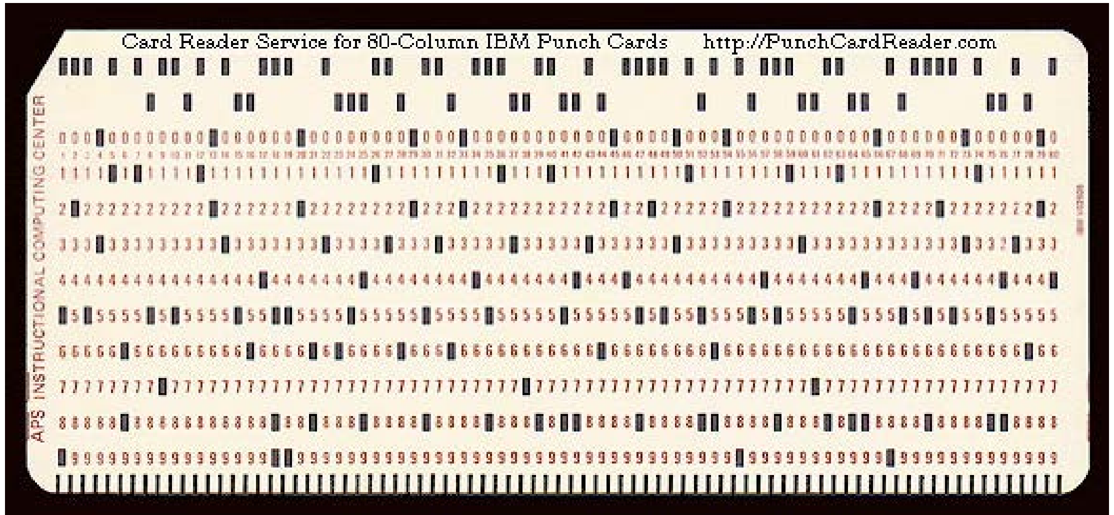
Space-delimited data, multiple lines
second part of read_spaced_data.py

```
def process_file(reader):
    """ (file open for reading) -> int
    Read and process reader, which must start with a time_series header.
    Return the largest value after the header. There may be multiple pieces
    of data on each line.
    """

    line = time_series_read.skip_header(reader).strip()

    # The largest value so far is the largest on this first line of data.
    largest = find_largest(line)

    # Check the rest of the lines for larger values.
    for line in reader:
        large = find_largest(line)
        if large > largest:
            largest = large

    return largest

if __name__ == '__main__':
    with open('lynx.txt', 'r') as input_file:
        print(process_file(input_file))
```

Read a punch card file?

Utermohle data:

```
1AB    432 1       182179 99130132110   11013212176104 997211623634726133655
2AB    432 1       435236420736 99   981510011190805020311227521092351 93314552
3AB    432 1       1411610341113 91101   95 82 82 78 72 76
1AB    182 1       194190108143141117   119137128841141097212324625122124054
2AB    182 1       425236430738100211031910312210906020512229601132053104315961
3AB    182 1         1301123913104113103    93 87 85 81 84
1CG2426921 1       182178103139137117   12114213279112 996812521664326133959
2CG2426921 1       50523540094110117 9513 9705201003010311228541102257 97264347
3CG2426921 1       191201034818 94102 97102 86 84 83 78 81
1CG2428341 1       179173102135132115   11914112377112 997212325705028113958
2CG2428341 1       41513340093610122 9913 9907201202010410524471072649 96304753
3CG2428341 1       191151004420 91103 97 99 79 81 78 74 77
```


Card Reader Service for 80-Column IBM Punch Cards
http://PunchCardReader.com
APS INSTRUCTIONAL COMPUTING CENTER

Homework 11

Due 10/25 before class

Exercises 10.10

1, 2,

A. Using for and while, write a function that will warn if a file exists, ask the user if he/she wants to choose another name, overwrite, or simply cancel. Of course, any new name must be tested too (while). You have most of the necessary code on slide 7 and 8. This will be part of your toolkit.

and...

Homework 11 continued:

B. Read one format into another

Hanihara data: (save to a text file)

```
Specimen 1
  182.00  179.00  100.00  129.00   95.00  108.00  115.00  114.00  100.00  132.00
  130.00  134.00  103.00  113.00  120.00   88.00  105.00  107.00  125.00   94.00
   65.00   23.00   44.00   41.00   37.00   27.00   50.00   49.00   71.00   31.00
   20.00  109.00   83.00  110.00   30.00   12.00   35.00   61.00   35.00  101.00
   54.00   54.00   11.12    5.70    6.89   95.00   52.00   54.00
Specimen 2
  174.00  172.00   96.00  124.00   95.00  110.00  104.00  103.00   96.00  137.00
  127.00  125.00  108.00  112.00  112.00   95.00   93.00  102.00    0.00   86.00
   61.00   21.00   41.00   39.00   35.00   25.00   50.00   49.00   62.00   25.00
   17.00   99.00   83.00  100.00   27.00   11.00   30.00   50.00   31.00   94.00
   49.00   51.00    8.73    0.00    0.00   84.00   47.00   47.00
Specimen 3
  170.00  167.00   92.00  130.00   93.00  109.00  116.00  115.50  100.00  130.00
  123.00  123.00  110.00  107.00  109.00   92.00   93.00  102.00  125.00   87.00
   65.00   21.00   40.00   38.00   34.00   24.00   45.00   45.00   61.00   19.00
   13.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   91.60
   49.10   48.70    7.76    5.22    3.97   89.20   48.60   49.70
```

Convert to .csv format

Read one format into another

Hanihara data format:

Specimen 1 <- specimen number always less than 30 characters.

48 measurements (mostly integers)

0.00 = missing (change to 'NA')

The first line of the converted .csv file will be (all one line):

```
"HSpecNo","GOL","NOL","BNL","XCB","M9","XFB","M11","AUB","ASB","BBH","M26","M27","M28","FRC",
"PAC","OCC","BPL","M43","ZYB","M46","NPH","DKB","M51","OBH","OBH","NLB","NLH","M55","MAB",

"MDH","MDB","U1","U2","U3","U4","U5","U6","U7","U8","U9","U10","U11","WNB","SIS","U12","ZMB",
"U13","U14"
```

- then append the data from the records into the file delimited using commas.

File example:

```
"HSpecNo","GOL","NOL","BNL","XCB"," ...

"Specimen 1",182.00,179.00, 100.00, 129.00, 95.00, 108.00, ...
```

Submit code and file. Think about helper functions (part of a toolkit).