Reading Data From a File

- Suppose the file named *data1.txt* contains the following numbers:

```
21.8
18.1
19
23
26
17.8
```

- We wish to use these numbers in a program.

- Read and print each line in the file. (readData1.py)

```
from time import ctime

def main():
    infile = open('data1.txt', 'r') # 'r' means
read
    for line in infile:
        print(line, end='')
    infile.close()

    print("""\n
Programmed by Stew Dent.
Date: %s.
End of processing.""" % ctime())

main()
```

- The output from the program is:

```
21.8
18.1
19
23
26
17.8

Programmed by Stew Dent.
Date: Sat Jun 30 08:09:04 2018.
End of processing.
```

- In order to read data from a file you must ***open*** the file by specifying the name of the file as a string and the operation as 'r'.
- After the file has been opened you can read from the file.

  e.g. `infile = open('data1.txt', 'r')`
- When you have finished using the file you should close the file.

  e.g. `infile.close()`
- The actual reading of the data is done in the statement

  `for line in infile:`
- The loop is executed once for each line in the file.
- The data in the line of the file is assigned to the variable ***line*** as a **string** value.

•Read all of the lines from the file into a list of strings. (readData2.py)

```
from time import ctime

def main():
    infile = open('data1.txt', 'r')
    lines = infile.readlines()
    infile.close()

    print(lines)
    for line in lines:
        print(line, end='')

    print("""\n
Programmed by Stew Dent.
Date: %s.
End of processing.""" % ctime())

main()
```

•An alternate way to read the lines into a list of strings.

```
# readData3.py

from time import ctime

def main():
    infile = open('data1.txt', 'r')
    lines = [line for line in infile]
    infile.close()
    print(lines)
    for line in lines:
        print(line.strip())
    print("""\n
Programmed by Stew Dent.
Date: %s.
End of processing.""" % ctime())

main()
```

•The output from these programs is:

```
['21.8\n', '18.1\n', '19\n', '23\n', '26\n',
'17.8']
21.8
18.1
19
23
26
17.8

Programmed by Stew Dent.
Date: Sat Jun 30 09:12:23 2018.
End of processing.
```

•Notice that the lines are separated by newline characters.

•Compute the average of the numbers that have been read. To do this each string must be converted to a float. (readAverage1.py)

```
def main():
    infile = open('data1.txt', 'r')
    lines = infile.readlines()
    infile.close()

    total = 0.0
    for line in lines:
        print(line, end='')
        total += float(line)
    print('\nThe average is', total /
          len(lines))

main()
```

- The output from this program is:

```
21.8
18.1
19
23
26
17.8
The average is 20.95
```

•The average can also be computed as follows.
(readAverage2.py)

```
def main():
    infile = open('data1.txt', 'r')
    lines = infile.readlines()
    infile.close()

    total = sum([float(line) for line in lines])
    print('\nThe average is', total / len(lines))

main()
```

•The output from the program is:

```
The average is 20.95
```

•The average can also be computed as follows (readAverage3.py)

```
def main():
    infile = open('data1.txt', 'r')
    numbers = [float(line) for line in infile]
    infile.close()

    average = sum(numbers) / len(numbers)
    print('\nThe average is', average)

main()
```

•The output from the program is:

```
The average is 20.95
```

•The function ***readline*** returns the empty string when the end of the file is reached.

```
In [1]: xxx = ""
In [2]: if xxx: print(True)
        else: print(False)
False
In [3]: xxx = 'hello'
In [4]: if xxx: print(True)
        else: print(False)
True
```

•As the empty string is treated as False in a condition and a non-empty string is treated as True in a condition the result of *readline* can be used to control a *while* loop.

•Read and display each line of a file: (readFile.py)

```
def main():
    infile = open('data1.txt', 'r')
    print('The data in the file is:')
    data = infile.readline()
    while data:
        print(data, end='')
        data = infile.readline()
    infile.close()

main()
```

- The output from the program is:

```
The data in the file is:
21.8
18.1
19
23
26
17.8
```

•The function ***read*** returns the contents of the entire file as a string. (readFile1.py)

```
def main():
    infile = open('data1.txt', 'r')
    print 'The data in the file is:'
    data = infile.read()
    print(data)
    infile.close()

main()
```

- The output from the program is:

```
The data in the file is:
21.8
18.1
19
23
26
17.8
```

Strings

- A string is zero or more characters enclosed in either single or double quotes.
- A string is of type ***str***.
- e.g.

  "" is the empty string which contains no characters.

  'hello'

  "I'm home!"

  'I\'m home!'

- If a quote needs to be embedded within a string use the opposite quote to enclose the string or precede the quote with the escape character \
- To include a \ in a string use \\.

•If s is a string then s[i:j] is the ***substring*** starting at position i and ending at position j-1

```
In [1]: s = 'My hair is red.'
In [2]: s[0:2]
'My'
In [3]: s[3:len(s)] # goes to the end of the string
'hair is red.'
In [4]: s[3:-1]     # excludes the last character
'hair is red'
```

•s.find(s1)

–returns the position of the first occurrence of s1 within s

–if s1 does not occur within s returns -1

```
In [5]: print(s.find('hair'))
3
In [6]: print(s.find('Joe'))
-1
```

- Determine if one string occurs within another string.

```
In [1]: s = 'My hair is red.'
In [2]: print('red' in s)
True
In [3]: print('umbrella' in s)
False
```

- Consider the following:

```
In [4]: print(s.startswith('My'))
True
In [5]: print(s.startswith('house'))
False
```

- Consider the following:

```
In [6]: s = 'This is my house.'
In [7]: s.replace('is', 'at')
'That at my house.'
```

- Replaces every occurrence of ***is*** with ***at***.

- You can specify the number of occurrences to be replaced.

```
In [1]: s = 'This is my house.'
In [2]: s.replace('is', 'at', 1)
'That is my house.'
```

- If s is a string then s.split() splits s into a list of tokens that are separated by whitespace (space, tab, newline).

```
In [3]: s = 'hello there\tmy\nman.'
In [4]: s.split()
['hello', 'there', 'my', 'man.']
```

- You can specify the separator (delimiter) split will use.

```
In [5]: s = 'cooper:1012:randy::/users/cooper'
In [6]: s.split(':')
['cooper', '1012', 'randy', '',
'/users/cooper']
```

- An empty string appears where two adjacent delimiters occur in s.

- If s is a string s.splitlines() splits a string into a list of lines where each line is separated by a newline '\n' character.
- This is useful when a file has been read into a string and the string must be split into lines.

```
In [1]: s = 'line 1\nline 2\nline 3\n'
In [2]: s.splitlines()
['line 1', 'line 2', 'line 3']
```

- Let s be a string variable.
- s.lower() creates and returns a new string in which all of the uppercase letters have been converted to lowercase. None of the characters in **s** are changed.

```
In [3]: s = 'HELLO 123 abc'
In [4]: s.lower()
'hello 123 abc'
In [5]: s
'HELLO 123 abc'
```

•s.upper() creates and returns a new string in which all of the lowercase letters have been converted to uppercase. None of the characters in **s** are changed.

```
In [1]: s = 'HELLO 123 abc'
In [2]: s.upper()
'HELLO 123 ABC'
In [3]: s
'HELLO 123 abc'
```

•Strings are constants and characters within the string cannot be changed.

```
In [4]: s = 'hello'
In [5]: s[0] = 'H'

-----> 1 s[0] = 'H'
TypeError: 'str' object does not support item assignment
```

•Assigning a new string to a variable that already holds a string is not changing a string.

```
In [1]: s = 'hello'
In [2]: s = 'Hello'  # This is valid!
```

•To replace characters within a string assigned to a variable build a new string with the desired value and assign it to the variable.

```
In [3]: s = 'My hair is red.'
In [4]: s = s[:3] + 'car' + s[7:]
In [5]: s
'My car is red.'
```

String Contents

- Let s be a string variable.
- *s.isdigit()* is True if all of the characters in the string s are decimal digits, otherwise it is False.
- *s.isalpha()* is True if all of the characters in the string s are letters, otherwise it is False (a space is not considered to be a letter).
- *s.isalnum()* is True if all of the characters in the string s are letters or digits, otherwise it is False.
- *s.isupper()* is True if all of the letters in the string s are upper case, otherwise it is False.
- *s.islower()* is True if all of the letters in the string s are lower case, otherwise it is False.
- *s.isspace()* is True if all of the characters in the string s are whitespace, otherwise it is False (whitespace is space, tab, newline).

•e.g. test for a blank line

```
In [1]: line = '      \t       \n'
In [2]: print(line.isspace())
True
In [3]: print(line.strip() == "")
True
```

Variations of strip()

```
In [4]: line = '\t  hello   there   \n'
In [5]: line.strip() #strip on left & right
ends
'hello   there'
In [6]: line.lstrip() # strip from left end
only
'hello   there   \n'
In [7]: line.rstrip() # string from right end
'\t  hello   there'
```

Joining Strings

•Opposite of split, joins together a list of strings.

e.g.

```
In [1]: s = 'My hair is red.'
In [2]: t = s.split(' ')
In [3]: t
['My', 'hair', 'is', 'red.']
In [4]: s1 = ' '.join(t)
In [5]: s1
'My hair is red.'
```

•*split* and *join* must use the same delimiter, which in this case is ' ', in order for s1 to be the same as s.

•Remove the first and last words / tokens from a string.

```
In [1]: s = 'Joe Smith is my name eh'
In [2]: t = s.split()
In [3]: t
['Joe', 'Smith', 'is', 'my', 'name', 'eh']
In [4]: s1 = ' '.join(t[1:-1])
In [5]: s1
'Smith is my name'
```

•An example from the old textbook. Read a file where each line is of the form: (x_0,y_0) (x_1,y_1) (x_2,y_2) … and ends with a newline character.

•Create a list of tuples of the form: $[(x_0,y_0), (x_1,y_1), \ldots, (x_n,y_n)]$ where x_i and y_i are real numbers (floats). `(readCoords.py)`

```
from time import ctime
def getCoords(filename):
    infile = open(filename, 'r')
    lines = infile.readlines()
 infile.close()
    coords = []
    for line in lines:
        points = line.strip().split() # remove \n
        for point in points:
            point = point[1:-1] # remove ( )
            coord = point.split(',')
            n = (float(coord[0]) ,
                 float(coord[1]))
            coords.append(n)
 return coords
```

```
def displayCoords(coords):
    print('   X\t   Y'
    for x, y in coords:
        print '%6.2f\t%6.2f' % (x, y))

def main():
    print '-------------------------------
\n'
    displayCoords(getCoords('coords1.txt'))
    print("""
Programmed by Stew Dent.
Date: %s
End of processing.""" % ctime())

main()
```

•The output from the program is:

```
-------------------------------------------------

   X          Y
  1.30         0.00
 -1.00         2.00
  3.00        -1.50
  0.00         1.00
  1.00         0.00
  1.00         1.00
  0.00        -0.01
 10.50        -1.00
  2.50        -2.50

Programmed by Stew Dent.
Date: Sat Jun 30 11:12:12 2018
End of processing.
```