7.3 Reading Files

To open a file for reading you must use

```
F = open(<file name>, "r")
```

The file must exist and be in the right folder; it is an error to try to open a file that doesn't exist in the folder where the system expects to find it. For what we are doing now, the system looks for the file in the folder that contains the program. So until we talk about moving around in a file system, you should place any file your program will read in the same folder as the program itself. If you don't, your program will crash with a "File not found" exception.

If F is an open file – i.e., if it is the value returned by a call to **open**, there are several ways to read data from F:

F.read() returns the entire file as one long string, using \n characters for line breaks

F.read(n) returns the next **n** characters from the file as a string.

F. readline () returns the next line of the file as a string.

F. readlines () returns all of the file as a list of strings, each string representing one line of the file.

You can also read the entire file with a for-loop:

```
for line in F:
    ...
```

In most situations it is easiest to read through the whole file with a **for**-loop. Text files have a special end-of-line character that marks the end of each line; this is usually denoted "\n". When you iterate through a text file with a **for**-loop, this end-of-line character remains attached to the lines; your first step is usually to eliminate it:

```
for line in F:
        line = line.strip("\n")
        ...
```

After that, the line is a standard string that you can treat the way you would treat any string.

Here, for example, is a program that will print its own text to the screen.

```
# This program prints the file "Program7.3.1.py",
# which happens to contain the text of this
# program.

def main():
    F = open( "Program7.3.1.py", "r" )
    for line in F:
        line = line.strip("\n")
        print(line)

main()
```

Program 7.3.1: This prints its own text

Next, we have a program that reads a column of numbers and prints its average. As with many situations when working with files, we dump the file of numbers into a list, and then use our previous techniques to find the average of the list. Note that the lines of the file are strings, so we need to convert these lines into numbers before we average them.

```
def Average(L):
    sum = 0
    count = 0
    for x in L:
        count = count + 1
        sum = sum + x
    return sum/count

def main():
    F = open( "nums.txt", "r")
    numbers = []
    for line in F:
        x = int(line)
        numbers.append(x)
    print( "I count %d numbers." % len(numbers) )
    print( "Their average is %.2f" % Average(numbers) )

main()
```

Program 7.3.2: This averages the numbers in a data file.

If this program sees a file "nums.txt" containing the numbers

```
2
9
4
3
5
6
```

it outputs:

```
I count 6 numbers.
Their average is 4.83
```

As a final example, here is a similar program that reads a data file with two columns of numbers, such as

```
3   29
4   45
7   52
5   43
```

```
def Avg(L):
    sum = 0
    count = 0
    for x in L:
        count = count + 1
        sum = sum + x
    return sum/count

def main():
    F = open( "2cols.txt", "r")
    column1 = []
    column2 = []
    for line in F:
        nums = line.split()
        x = int(nums[0])
        y = int(nums[1])
        column1.append(x)
        column2.append(y)
    avg1 = Avg(column1)
    avg2 = Avg(column2)
    print( "I count %d rows." % len(column1) )
    print( "The averages were %.2f and %.2f" % (avg1,avg2) )

main()
```

Program 7.3.3: This averages two columns of data from a file.