Download this page as a jupyter notebook at Lesson 7

ENGR 1330 Computational Thinking with Data Science

Last GitHub Commit Date: 10 September 2021

Lesson 7 Files:

- Files and Filesystems
- Opening and reading text (ascii) files
- Using loops to read text files
- Writing to text files
- Download files from a remote server
- Creating and deleting files from your notebook

In [2]:
```
# Script block to identify host, user, and kernel
import sys
! hostname; ! whoami; ! pwd;
print(sys.executable)
```

```
atomickitty
sensei
/home/sensei/engr-1330-webroot/1-Lessons/Lesson07
/opt/jupyterhub/bin/python3
```

In [3]:
```
%%html
<!-- Script Block to set tables to left alignment -->
<style>
  table {margin-left: 0 !important;}
</style>
```

Objectives

1. Describe files, file types, and search paths
2. Demonstrate file creation, and deletion in a script
3. Demonstrate create, open, read from a file
4. Demonstrate open, write to a file
5. Apply packages to directly obtain a file from a remote server

Files and Filesystems

A computer file is a computer resource for recording data discretely (not in the secretive context, but specifically somewhere on a piece of hardware) in a computer storage device. Just as words can be written to paper, so can information be written to a computer file. Files can be edited and transferred through the internet on that particular computer system.

There are different types of computer files, designed for different purposes. A file may be designed to store a picture, a written message, a video, a computer program, or a wide variety of other kinds of data. Some types of files can store several types of information at once.

By using computer programs, a person can open, read, change, save, and close a computer file. Computer files may be reopened, modified, and copied an arbitrary number of times.

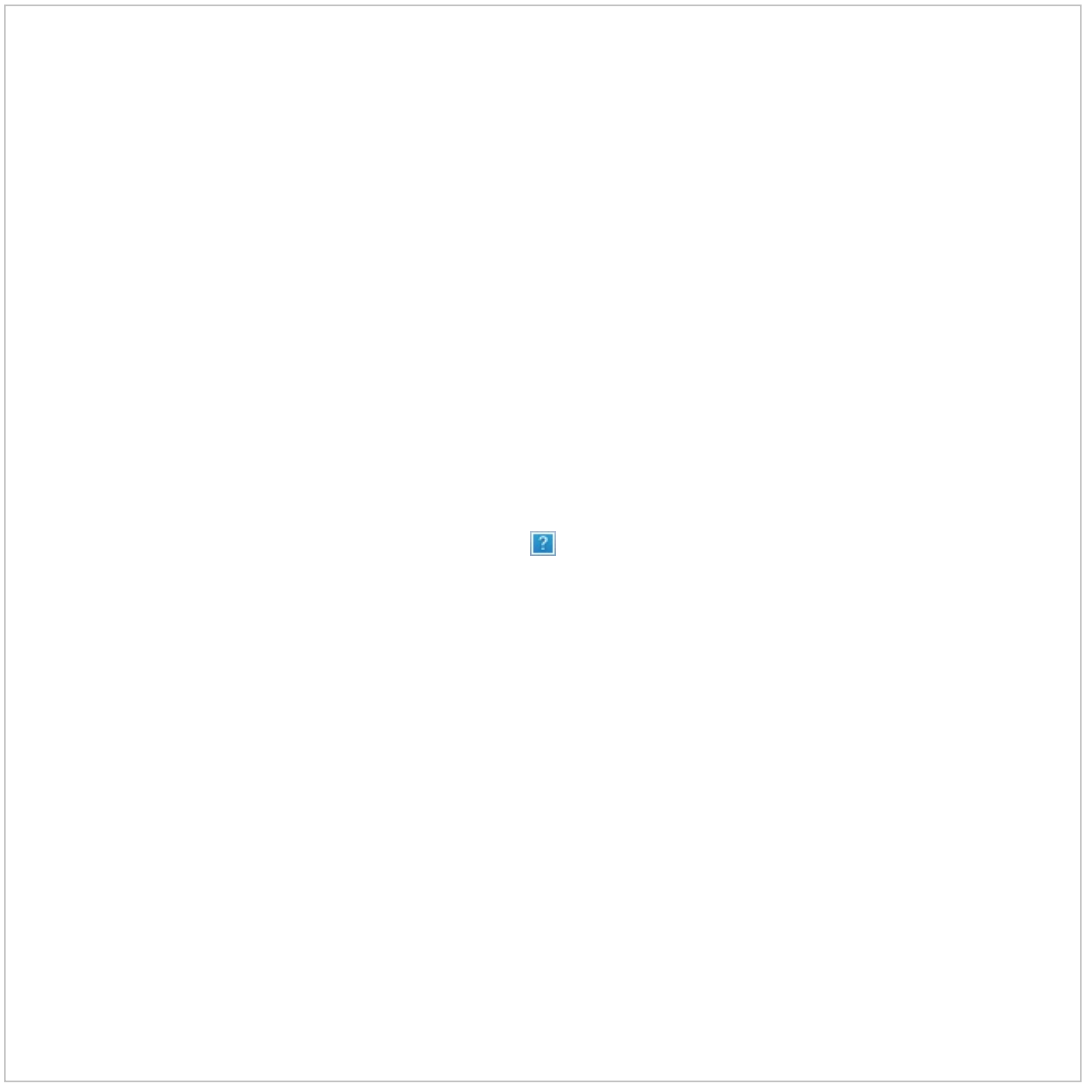
Typically, files are organised in a file system, which keeps track of where the files are located on disk and enables user access.

File system

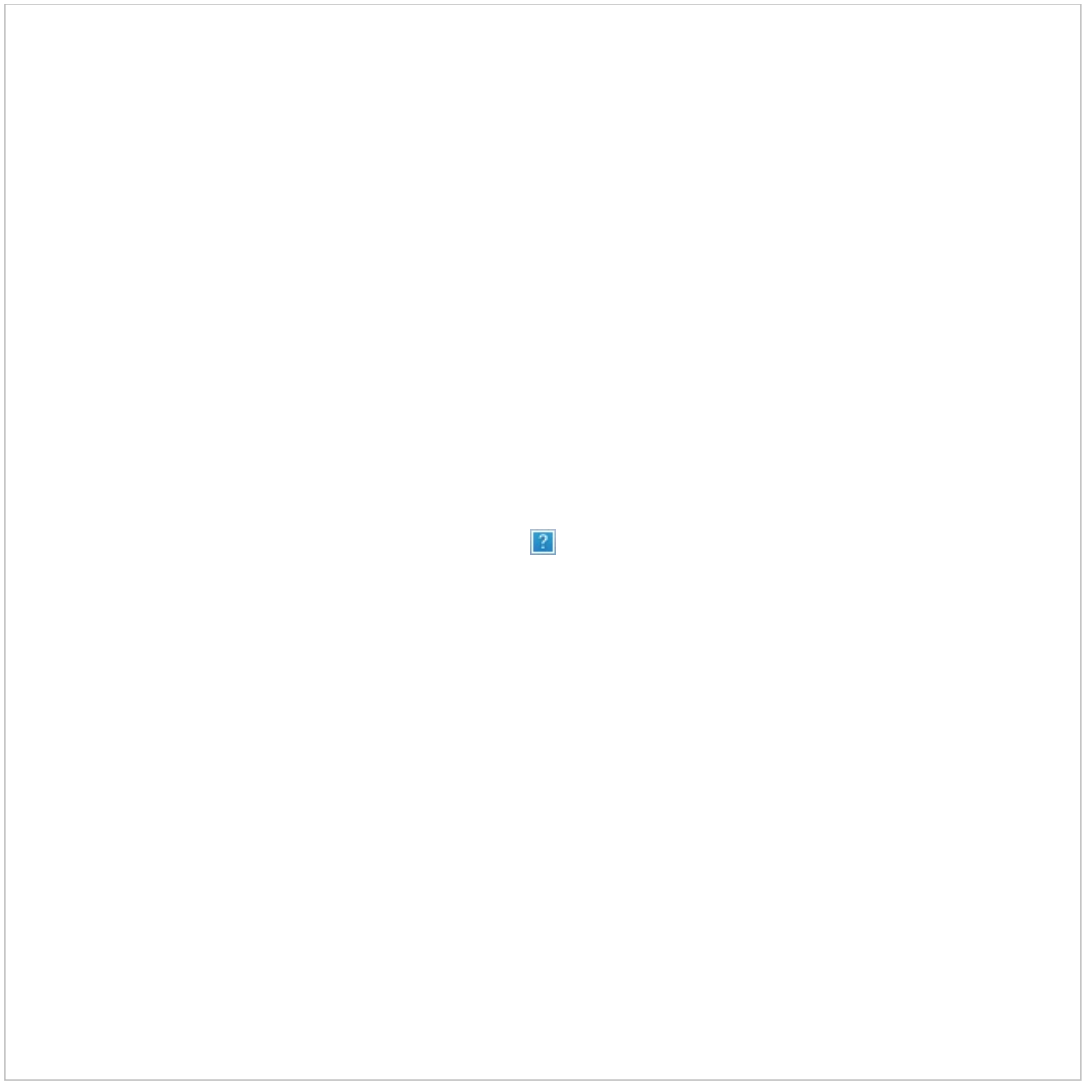
In computing, a file system or filesystem, controls how data is stored and retrieved. Without a file system, data placed in a storage medium would be one large body of data with no way to tell where one piece of data stops and the next begins. By separating the data into pieces and giving each piece a name, the data is isolated and identified. Taking its name from the way paper-based data management system is named,

each group of data is called a “file”. The structure and logic rules used to manage the groups of data and their names is called a “file system”.

The figure below is a graphical representation of the filesystem on my office computer. I have the file browser listing the contants of a directory named `/src`. It is contained in a directory named `/final_report`, which is contained in a higher level directory all the way up to `/Users` (`/` is aliased to `Macintosh HD` in the figure)



An equivalent representation, in a `bash` shell is shown in the figure below which is a capture of a terminal window.



Path

A path, the general form of the name of a file or directory, specifies a unique location in a file system. A path points to a file system location by following the directory tree hierarchy expressed in a string of characters in which path components, separated by a delimiting character, represent each directory. The delimiting character is most commonly the slash (`/`), the backslash character (`\`), or colon (`:`), though some operating systems may use a different delimiter. Paths are used extensively in computing to represent the directory/file relationships common in modern operating systems, and are essential in the construction of Uniform Resource Locators (URLs). Resources can be represented by either absolute or relative paths. As an example consider the following two files:

1. /Users/theodore/MyGit/@atomickitty/hurri-sensors/.git/Guest.conf
2. /etc/apache2/users/Guest.conf

They both have the same file name, but are located on different paths. Failure to provide the path when addressing the file can be a problem. Another way to interpret is that the two unique files actually have different names, and only part of those names is common (Guest.conf) The two names above (including the path) are called fully qualified filenames (or absolute names), a relative path (usually relative to the file or program of interest depends on where in the directory structure the file lives. If we are currently in the .git directory (the first file) the path to the file is just the filename.

File Types

1. Text Files. Text files are regular files that contain information readable by the user. This information is usually stored as ASCII with various encoding extensions (to handle other alphabets and emojis). You can display and print these files. The lines of a text file must not contain NULL characters, and none can exceed a prescribed (by architecture) length, including the new-line character. The term text file does not prevent the inclusion of control or other nonprintable characters (other than NUL). Therefore, standard utilities that list text files as inputs or outputs are either able to process the special characters gracefully or they explicitly describe their limitations within their individual sections.

2. Binary Files. Binary files are regular files that contain information readable by the computer. Binary files may be executable files that instruct the system to accomplish a job. Commands and programs are stored in executable, binary files. Special compiling programs

translate ASCII text into binary code. The only real difference between text and binary files is that text files have lines of some prescribed length, with no NULL characters, each terminated by a new-line character.

3. Directory Files. Directory files contain information the system needs to access all types of files, but they do not contain the actual file data. As a result, directories occupy less space than a regular file and give the file system structure flexibility and depth. Each directory entry represents either a file or a subdirectory. Each entry contains the name of the file and the file's index node reference number (i-node). The i-node points to the unique index node assigned to the file. The i-node describes the location of the data associated with the file. Directories are created and controlled by a separate set of commands.

File Manipulation

For this lesson we examine just a handfull of file manipulations which are quite useful. Files can be "created","read","updated", or "deleted" (CRUD).

Example: Examine our local directory

First will use some system commands to view the contents of the local directory.

> The code blocks below are for Linux systems, and the instructions may not execute correctly on a windows machine

In [4]:
```
import sys
! rm -rf myfirstfile.txt # delete file if it exists
! pwd # list name of working directory, note it includes path, so it is an absolute path
```

```
/home/sensei/engr-1330-webroot/1-Lessons/Lesson07
```

In [5]:
```
! ls -l # list contents of working directory
```

```
total 548
-rw-rw-r-- 1 sensei sensei 122198 Sep 10 20:57 ENGR-1330-Lesson07.ipynb
-rw-rw-r-- 1 sensei sensei 343800 Sep 10 20:31 Filesystem-graphic.png
-rw-rw-r-- 1 sensei sensei  88568 Sep 10 20:31 Filesystem-shell.png
drwxr-xr-x 3 sensei sensei   4096 Jul 21 16:15 OriginalPowerpoint
```

Example: Create a file, write to it.

Below is an example of creating a file that does not yet exist. The script is a bit pendandic on purpose.

In [6]:
```
# create file example
externalfile = open("myfirstfile.txt",'w') # create connection to file, set to write (w), file does not need to e
mymessage = 'message in a bottle' #some object to write, in this case a string
externalfile.write(mymessage)# write the contents of mymessage to the file
externalfile.close() # close the file connection
```

At this point our new file should exist, lets list the directory and see if that is so

In [7]:
```
! ls -l # list contents of working directory
```

```
total 552
-rw-rw-r-- 1 sensei sensei 122198 Sep 10 20:57 ENGR-1330-Lesson07.ipynb
-rw-rw-r-- 1 sensei sensei 343800 Sep 10 20:31 Filesystem-graphic.png
-rw-rw-r-- 1 sensei sensei  88568 Sep 10 20:31 Filesystem-shell.png
drwxr-xr-x 3 sensei sensei   4096 Jul 21 16:15 OriginalPowerpoint
-rw-rw-r-- 1 sensei sensei     19 Sep 10 20:58 myfirstfile.txt
```

Sure enough, its there,

> We will use a Linux shell command `cat` to look at the contents of the file.

In [8]:
```
! cat myfirstfile.txt
```

```
message in a bottle
```

Example: Read from an existing file.

We will continue using the file we just made, and read from it the example is below

In [6]:
```
# read file example
externalfile = open("myfirstfile.txt",'r') # create connection to file, set to read (r), file must exist
silly_string = externalfile.read() # read the contents
externalfile.close() # close the file connection
print(silly_string)
```

```
message in a bottle
```

These first two examples are supremely simple, but illustrate important points:

1. The program establishes a connection to the file
2. One connection exists, then read from, write to the file
3. Close the connection (to clear the I/O buffer and release the connection thread)

In [9]:
```
# write without a connection -- observe the error message
mymessage = 'message in a bottle' #some object to write, in this case a string
externalfile.write(mymessage)# write the contents of mymessage to the file
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-81084545fc2e> in <module>
      1 # write without a connection
      2 mymessage = 'message in a bottle' #some object to write, in this case a string
----> 3 externalfile.write(mymessage)# write the contents of mymessage to the file

ValueError: I/O operation on closed file.
```

In [10]:
```
# read without a connection -- observe the error message
silly_string = externalfile.read() # read the contents
print(silly_string)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-38ae4e51ad51> in <module>
      1 # read without a connection -- observe the error message
----> 2 silly_string = externalfile.read() # read the contents
      3 print(silly_string)

ValueError: I/O operation on closed file.
```

Example: Update a file.

This example continues with our same file, but we will now add contents without destroying existing contents. The keyword is `append`

In [11]:
```
externalfile = open("myfirstfile.txt",'a') # create connection to file, set to append (a), file does not need to
externalfile.write('\n') # adds a newline character
what_to_add = 'I love rock-and-roll, put another dime in the jukebox baby ... \n'
externalfile.write(what_to_add) # add a string including the linefeed
what_to_add = '... the waiting is the hardest part \n'
externalfile.write(what_to_add) # add a string including the linefeed
mylist = [1,2,3,4,5] # a list of numbers
what_to_add = ','.join(map(repr, mylist)) + "\n" # one way to write the list
externalfile.write(what_to_add)
what_to_add = ','.join(map(repr, mylist[0:len(mylist)])) + "\n" # another way to write the list
externalfile.write(what_to_add)
externalfile.close()
```

In [12]:
```
externalfile = open("myfirstfile.txt",'a') # create connection to file, set to append (a), file does not need to
externalfile.write('\n') # adds a newline character
what_to_add = 'I love rock-and-roll, put another dime in the jukebox baby ... \n'
externalfile.write(what_to_add) # add a string including the linefeed
externalfile.close()
```

As before we can examine the contents using a shell command sent from the notebook.

In [13]:
```
! cat myfirstfile.txt
```

```
message in a bottle
I love rock-and-roll, put another dime in the jukebox baby ...
... the waiting is the hardest part
1,2,3,4,5
1,2,3,4,5
```

```
I love rock-and-roll, put another dime in the jukebox baby ...
```

Example: Delete a file

Delete can be done by a system call as we did above to clear the local directory

In a JupyterLab notebook, we can either use

```
import sys
! rm -rf myfirstfile.txt # delete file if it exists
```

or

```
import os
os.remove("myfirstfile.txt")
```

they both have same effect, both equally dangerous to your filesystem.

Learn more about CRUD with text files at https://www.guru99.com/reading-and-writing-files-in-python.html

Learn more about file delete at https://www.dummies.com/programming/python/how-to-delete-a-file-in-python/

In [24]:
```
import os
file2kill = "myfirstfile.txt"
try:
    os.remove(file2kill) # file must exist or will generate an exception
except:
    pass # example of using pass to improve readability
print(file2kill, " missing or deleted !")
```

```
myfirstfile.txt  missing or deleted !
```

A little discussion on the part where we wrote numbers

```
what_to_add = ','.join(map(repr, mylist[0:len(mylist)])) + "\n"
```

Here are descriptions of the two functions `map` and `repr`

`map(function, iterable, ...)` Apply `function` to every item of iterable and return a list of the results. If additional iterable arguments are passed, function must take that many arguments and is applied to the items from all iterables in parallel. If one iterable is shorter than another it is assumed to be extended with None items. If function is None, the identity function is assumed; if there are multiple arguments, `map()` returns a list consisting of tuples containing the corresponding items from all iterables (a kind of transpose operation). The iterable arguments may be a sequence or any iterable object; the result is always a list.

`repr(object)` Return a string containing a printable representation of an object. This is the same value yielded by conversions (reverse quotes). It is sometimes useful to be able to access this operation as an ordinary function. For many types, this function makes an attempt to return a string that would yield an object with the same value when passed to `eval()`, otherwise the representation is a string enclosed in angle brackets that contains the name of the type of the object together with additional information often including the name and address of the object. A class can control what this function returns for its instances by defining a `repr()` method.

What they do in this script is important. The statement:

```
what_to_add = ’,’.join(map(repr, mylist[0:len(mylist)])) + "\n"
```

is building a string that will be comprised of elements of mylist[0:len(mylist)]. The `repr()` function gets these elements as they are represented in the computer, the delimiter a comma is added using the join method in Python, and because everything is now a string the

```
... + "\n"
```

puts a linefeed character at the end of the string so the output will start a new line the next time something is written.

Example

- create a text file, name it **"MyFavoriteQuotation"**.
- Write **your favorite quotation** in the file.
- Read the file.
- Add this string to it in a new line : "And that's something I wish I had said..."
- Show the final outcome.

In [15]:
```
# create the "My Favorite Quotation" file:
externalfile = open("MyFavoriteQuotation.txt",'w')         # create connection to file, set to write (w)
myquotation = 'The path of the righteous man is beset on all sides by the inequities of the selfish and the tyran
externalfile.write(myquotation)# write the contents of mymessage to the file
externalfile.close() # close the file connection
#Let's read the file
! cat MyFavoriteQuotation.txt
# Let's add the string
externalfile = open("MyFavoriteQuotation.txt",'a')  #create connection to file, set to append (a)
externalfile.write('\n') # adds a newline character
what_to_add = "And that's something I wish I had said ... \n"
externalfile.write(what_to_add)
externalfile.close()
#Let's read the file one last time
! cat MyFavoriteQuotation.txt
```

```
The path of the righteous man is beset on all sides by the inequities of the selfish and the tyranny of evil men.
Blessed is he who, in the name of charity and good will, shepherds the weak through the valley of darkness. For h
e is truly his brother’s keeper and the finder of lost children. And I will strike down upon thee with great veng
eance and furious anger those who attempt to poison and destroy my brothers. And you will know my name is the Lor
d when I lay my vengeance upon you.The path of the righteous man is beset on all sides by the inequities of the s
elfish and the tyranny of evil men. Blessed is he who, in the name of charity and good will, shepherds the weak t
hrough the valley of darkness. For he is truly his brother’s keeper and the finder of lost children. And I will s
trike down upon thee with great vengeance and furious anger those who attempt to poison and destroy my brothers.
And you will know my name is the Lord when I lay my vengeance upon you.
And that's something I wish I had said ...
```

Reading data from a file.

To continue our exploration, suppose we want to read from a file, and we know it is a data file - in this section the files we will use are `A.txt` , `B.txt` , and `x.txt` all located at http://54.243.252.9/engr-1330-webroot/4-Databases/ to follow along download these files to the directory where your script is running.

Our task is to determine if x is a solution to $A \cdot x = B$

From our problem solving protocol the algorithmic task is

> 1. Allocate objects to store file contents;
> 2. Read in A,B, and x from respective files;
> 3. Echo the inputs (pedagogical in this case);
> 4. Perform the matrix arithmetic Ax = RHS;
> 5. Test if RHS == B;
> 6. Report results;

In [203…
```
# Lets look at the files using shell commands (may need to adapt for windoze)
! echo '--- A.txt ---'
! cat A.txt
! echo
! echo '--- x.txt ---'
! cat x.txt
! echo
! echo '--- B.txt ---'
! cat B.txt
```

```
--- A.txt ---
4.0  1.5  0.7  1.2  0.5
1.0  6.0  0.9  1.4  0.7
0.5  1.0  3.9  3.2  0.9
0.2  2.0  0.2  7.5  1.9
1.7  0.9  1.2  2.3  4.9

--- x.txt ---
0.59519
0.50793
0.83171
0.63037
1.03738

--- B.txt ---
```

```
5.0
6.0
7.0
8.0
9.0
```

First we make room for the data

In [174…
```
# Code to read A, X, and b - Notice we need somewhere for the data to go, hence the null lists
amatrix = [] # null list to store matrix read
xvector = [] # null list to store vector read
bvector = [] # null list to store vector read
rowNumA = 0
colNumA = 0
rowNumB = 0
rowNumX = 0
```

Next read in A

In [175…
```
localfile = open("A.txt","r") # connect and read file for MATRIX A
for line in localfile:
    amatrix.append([float(n) for n in line.strip().split()])
    rowNumA += 1
localfile.close() # Disconnect the file
colNumA = len(amatrix[0]) # get the column count
```

Echo the input

In [176…
```
print('A matrix')
for i in range(0,rowNumA,1):
    print ( (amatrix[i][0:colNumA]))
```

```
A matrix
[4.0, 1.5, 0.7, 1.2, 0.5]
[1.0, 6.0, 0.9, 1.4, 0.7]
[0.5, 1.0, 3.9, 3.2, 0.9]
[0.2, 2.0, 0.2, 7.5, 1.9]
[1.7, 0.9, 1.2, 2.3, 4.9]
```

Next read in x

In [177…
```
localfile = open("x.txt","r") # connect and read file for VECTOR x
for line in localfile:
    xvector.append(float(line))  # vector read different -- just float the line
    rowNumX += 1
localfile.close() # Disconnect the file
```

Echo the input

In [178…
```
print('x vector')
for i in range(0,rowNumX,1):
    print ( (xvector[i]))
```

```
x vector
0.59519
0.50793
0.83171
0.63037
1.03738
```

Read in B

In [179…
```
localfile = open("B.txt","r") # connect and read file for VECTOR B
for line in localfile:
    bvector.append(float(line))  # vector read different -- just float the line
    rowNumB += 1
localfile.close() # Disconnect the file
```

Echo the input

In [180…
```
print('B vector')
for i in range(0,rowNumB,1):
    print ( (bvector[i]))
```

```
B vector
```

```
5.0
6.0
7.0
8.0
9.0
```

Now we need to perform the arithmetic.

In [204…
```
rhs = [0 for i in range(rowNumX)] # here we will store Ax = rhs
for i in range(0,rowNumA): # select row
    for j in range(0,colNumA): # dot product current row*xvector
            rhs[i]=rhs[i]+amatrix[i][j]*xvector[j]
for i in range(0,rowNumA,1): # print out the result
    print (rhs[i])
```

```
4.999986
5.999993
7.00002
8.000037
9.000025
```

Now we need to compare the results, visually they are close and the floating point arithmetic using truncated inputs cannot ever be exact, so lets use our selection methods; or just print them

In [207…
```
tolerance = 1.0e-04 # decide that 1 part in 10,000 is enough
same = True # here is our flag, as we compare element by element if we find one that is not close enough we quit
for i in range(0,rowNumA,1): # just march through the lists
    if abs(rhs[i]-bvector[i]) > tolerance: # too far apart
        same = False
        break # we can exit the loop,
    else:
        continue # keep checking
if same == True:
    print('The two vectors are the same, so x solves Ax=B')
else:
    print('The two vectors are different, so x does not solve Ax=B')

print('---Ax---','---B---')
for i in range(0,rowNumA,1):
    print (' ',round(rhs[i],3),'    ',round(bvector[i],3))
```

```
The two vectors are the same, so x solves Ax=B
---Ax--- ---B---
  5.0      5.0
  6.0      6.0
  7.0      7.0
  8.0      8.0
  9.0      9.0
```

Downloading files from websites (optional)

This section shows how to get files from a remote computer. In the previous example, we can avoid the tedious select-right-click-save target .... step. There are several ways to get files here we examine just one.

The most important part is you need the FQDN (URL) to the file.

A Method to get the actual file from a remote web server (unencrypted)

> - You know the FQDN to the file it will be in structure of "http://server-name/.../filename.ext"
> - The server is running ordinary (unencrypted) web services, i.e. `http://...`

We will need a module to interface with the remote server. Here we will use `requests` , so first we load the module

> You may need to install the module into your anaconda environment using the anaconda power shell, on my computer the commands are:
>
> - **sudo -H /opt/jupyterhub/bin/python3 -m pip install requests**
>
> Or:

> - **sudo -H /opt/conda/envs/python/bin/python -m pip install requests**
>
> You will have to do some reading, but with any luck something similar will work for you.

The example below will get a copy of a file named `all_quads_gross_evaporation.csv` that is stored on the class server, the FQDN/URL is http://54.243.252.9/engr-1330-webroot/4-Databases/all_quads_gross_evaporation.csv. Here we can observe that the website is unencrypted `http` instead of `https`. If we visit the URL we can confirm that the file exists (or get a 404 error, if there is no file).

First we will import the requests module.

In [45]:
```
import requests # Module to process http/https requests
```

Assuming the requests module loads, lets next clear any existing local (to our machine) copies of the file, in this example, we already have the name, so will just send a system command to delete the file. This step is mostly for the classroom demonstration - the script will happily clobber existing files.

> The system command below may be different on windoze! What's here works on MacOS and Linux.

In [46]:
```
import sys # Module to process commands to/from the OS using a shell-type syntax
! rm -rf all_quads_gross_evaporation.csv # delete file if it exists
```

Now we will generate a `GET` request to the remote http server. I chose to do so using a variable to store the remote URL so I can reuse code in future projects. The `GET` request (an http/https method) is generated with the requests method `get` and assigned to an object named `rget` -- the name is arbitrary. Next we extract the file from the `rget` object and write it to a local file with the name of the remote file - esentially automating the download process.

In [47]:
```
remote_url="http://54.243.252.9/engr-1330-webroot/4-Databases/all_quads_gross_evaporation.csv"  # set the url
rget = requests.get(remote_url, allow_redirects=True)  # get the remote resource, follow imbedded links
localfile = open('all_quads_gross_evaporation.csv','wb') # open connection to a local file same name as remote
localfile.write(rget.content) # extract from the remote the contents,insert into the local file same name
localfile.close() # close connection to the local file
```

In [50]:
```
# print(type(localfile)) # verify object is an I/O object
```

```
<class '_io.BufferedWriter'>
```

In [51]:
```
# verify file exists
! pwd # list absolute path to script directory
! ls -lah # list directory contents, owner, file sizes ...
```

```
/home/sensei/engr-1330-webroot/1-Lessons/Lesson07
total 924K
drwxr-xr-x  4 sensei sensei 4.0K Sep 12 17:20 .
drwxr-xr-x 28 sensei sensei 4.0K Aug 23 16:10 ..
drwxrwxr-x  2 sensei sensei 4.0K Sep 10 21:43 .ipynb_checkpoints
-rw-rw-r--  1 sensei sensei 121K Sep 12 17:20 ENGR-1330-Lesson07.ipynb
-rw-rw-r--  1 sensei sensei 336K Sep 10 20:31 Filesystem-graphic.png
-rw-rw-r--  1 sensei sensei  87K Sep 10 20:31 Filesystem-shell.png
drwxr-xr-x  3 sensei sensei 4.0K Jul 21 16:15 OriginalPowerpoint
-rw-rw-r--  1 sensei sensei 355K Sep 12 17:18 all_quads_gross_evaporation.csv
-rw-rw-r--  1 sensei sensei  206 Sep 10 21:08 myfirstfile.txt
```

Now we can list the file contents and check its structure, before proceeding.

In [86]:
```
! less all_quads_gross_evaporation.csv
```

```
:,104,105,106,107,108,204,205,206,207,208,304,305,306,307,308,309,404,405,406,407,408,409,410,411,412,413,414,504
,505,506,507,508,509,510,511,512,513,514,601,602,603,604,605,606,607,608,609,610,611,612,613,614,701,702,703,704,
705,706,707,708,709,710,711,712,713,714,803,804,805,806,807,808,809,810,811,812,813,814,907,908,909,910,911,912,1
008,1009,1010,1011,1108,1109,1110,1210^M1954-01,1.8,1.8,2.02,2.24,2.24,2.34,1.89,1.8,1.99,2.02,2.67,2.46,2.11,1.8
3,1.59,1.17,2.09,2.5,2.22,1.83,1.77,1.62,1.23,1.23,1.27,1.27,1.27,2.98,2.8,2.36,2.16,1.96,1.63,1.52,1.52,1.41,1.3
8,1.33,2.42,2.54,3.01,2.96,2.81,2.57,2.49,2.22,1.72,1.73,1.66,1.6,1.45,1.45,2.42,2.43,2.45,2.54,2.46,2.29,2.52,2.
17,1.78,2.19,2.08,1.87,1.37,1.39,2.45,2.25,2.05,2.3,2.41,2.02,1.94,2.45,1.85,1.53,1.27,1.26,1.93,1.9,2.37,1.91,1.
42,1.3,2.5,2.42,1.94,1.29,2.59,2.49,2.22,2.27^M1954-02,4.27,4.27,4.13,3.98,3.9,4.18,4.26,4.27,4.26,4.18,4.1,3.98,
3.8,3.9,4.69,3.81,3.83,3.48,3.34,3.24,4.16,3.68,3.13,4.22,4.21,2.53,2.39,4.39,4.01,3.99,4.52,5.01,4.69,4.47,4.34,
4.24,3.43,2.78,3.81,3.92,4.47,4.57,5.03,5.42,5.54,5.12,3.81,3.97,4.09,3.71,3.52,3.52,3.81,3.96,4.32,4.52,4.58,4.2
,4.28,3.63,3.02,3.66,3.56,3.45,3.33,3.38,4.32,4.16,4,4.01,3.87,3.73,3.64,3.73,3.17,2.76,2.86,3.07,3.25,3.61,3.83,
3.53,2.59,2.51,4.71,4.3,3.84,2.5,5.07,4.62,4.05,4.18^M1954-03,4.98,4.98,4.62,4.25,4.2,5.01,4.98,4.98,4.68,4.77,5.
45,4.93,4.75,4.36,4.91,3.92,5.2,4.7,4.33,3.83,4.73,4.3,4.56,4.94,4.82,3.07,2.93,6.31,5.58,4.82,5,5.77,5.46,5.36,5
.02,4.96,3.92,3.24,6,6.08,6.26,6.28,6.37,5.68,5.47,5.83,4.59,4.35,4.54,4.15,3.83,3.83,6,5.87,5.54,5.77,5.7,5.17,5
.39,4.7,3.96,4.05,4.02,4,3.81,3.83,5.54,5.21,4.88,5.85,5.94,4.87,4.42,4.1,3.7,3.36,3.26,3.49,5,4.81,4.78,4.32,3.2
1,3.21,6.21,6.06,5.02,3.21,6.32,6.2,5.68,5.7^M1954-04,6.09,5.94,5.94,6.07,5.27,6.31,5.98,5.89,5.72,5.22,6.84,6.13
,5.19,4.82,5.86,4.95,5.82,all_quads_gross_evaporation.csv
```

Structure kind of looks like a spreadsheet as expected; notice the unsuual character `^M`; this unprintable character is the *carriage return+line feed* control character for MS DOS (Windows) architecture. The script below will actually strip and linefeed correctly, but sometimes all that is needed is to make a quick conversion using the system shell.

> Here are some simple system commands (on a Linux or MacOS) to handle the conversion for ASCII files
>
> - `sed -e 's/$/\r/' inputfile > outputfile` # UNIX to DOS (adding CRs)
> - `sed -e 's/\r$//' inputfile > outputfile` # DOS to UNIX (removing CRs)
> - `perl -pe 's/\r\n|\n|\r/\r\n/g' inputfile > outputfile` # Convert to DOS
> - `perl -pe 's/\r\n|\n|\r/\n/g'   inputfile > outputfile` # Convert to UNIX
> - `perl -pe 's/\r\n|\n|\r/\r/g'   inputfile > outputfile` # Convert to old Mac

Now lets actually read the file into a list for some processing. We will read it into a null list, and split on the commas (so we will be building a matrix of strings). Then we will print the first few rows and columns.

In [89]:
```
# now lets process the file
localfile = open('all_quads_gross_evaporation.csv','r') # open a connection for reading
aList = [] # null list to store read
rowNumA = 0 # counter to keep track of rows,
for line in localfile:
    #aList.append([str(n) for n in line.strip().split()])
    aList.append([str(n) for n in line.strip().split(",")]) # process each line, strip whitespace, split on ","
    rowNumA += 1 # increment the counter
localfile.close() #close the connection - amatrix contains the file contents
# print((aList[0])) # print 1st row
for irow in range(0,10):
    print([aList[irow][jcol] for jcol in range(0,10)])  # implied loop constructor syntax
```

```
['YYYY-MM', '104', '105', '106', '107', '108', '204', '205', '206', '207']
['1954-01', '1.8', '1.8', '2.02', '2.24', '2.24', '2.34', '1.89', '1.8', '1.99']
['1954-02', '4.27', '4.27', '4.13', '3.98', '3.9', '4.18', '4.26', '4.27', '4.26']
['1954-03', '4.98', '4.98', '4.62', '4.25', '4.2', '5.01', '4.98', '4.98', '4.68']
['1954-04', '6.09', '5.94', '5.94', '6.07', '5.27', '6.31', '5.98', '5.89', '5.72']
['1954-05', '5.41', '5.09', '5.14', '4.4', '3.61', '5.57', '4.56', '4.47', '4.18']
['1954-06', '9.56', '11.75', '12.1', '9.77', '8.06', '9.47', '8.42', '8.66', '8.78']
['1954-07', '8.65', '11.12', '11.33', '11.12', '10.09', '9.44', '9.32', '9.42', '10.14']
['1954-08', '5.81', '7.68', '9.97', '11.34', '9.76', '7.15', '8.56', '8.59', '9.43']
['1954-09', '7.42', '10.41', '10.64', '8.68', '7.67', '7.39', '8.31', '8.65', '8.42']
```

Now suppose we are interested in column with the label 910, we need to find the position of the column, and lets just print the dates (column 0) and the evaporation values for cell 910 (column unknown).

We know the first row contains the column headings, so we can use a while loop to find the position like:

In [106…
```
flag = True
c910 = 0
while flag:
    try:
        if aList[0][c910] == '910': # test if header is 910
            flag = False # switch flag to exit loop
        else :
            c910 += 1 # increment counter if not right header
    except:
        print('No column position found, resetting to 0')
        c910 = 0
        break

if c910 != 0:
    for irow in range(0,10): # activate to show first few rows
#    for irow in range(0,rowNumA): # activate to print entire list
        print(aList[irow][0],aList[irow][c910])  # implied loop constructor syntax
```

```
YYYY-MM 910
1954-01 1.91
1954-02 3.53
1954-03 4.32
1954-04 4.51
1954-05 4.25
1954-06 6.85
1954-07 7.99
1954-08 7.88
1954-09 6.55
```

A Method to get the actual file from a remote web server (SSL/TLS encrypted)

> - You know the FQDN to the file it will be in structure of "https://server-name/.../filename.ext"
> - The server is running SSL/TLS web services, i.e. `https://...`
> - The server has a CA certificate that is valid or possibly a self-signed certificate

This section is saved for future semesters

References

1. Learn Python in One Day and Learn It Well. Python for Beginners with Hands-on Project. (Learn Coding Fast with Hands-On Project Book -- Kindle Edition by LCF Publishing (Author), Jamie Chan https://www.amazon.com/Python-2nd-Beginners-Hands-Project-ebook/dp/B071Z2Q6TQ/ref=sr_1_3?dchild=1&keywords=learn+python+in+a+day&qid=1611108340&sr=8-3

---

1. Read a file line by line https://www.geeksforgeeks.org/read-a-file-line-by-line-in-python/
2. Read a file line by line (PLR approach) https://www.pythonforbeginners.com/files/4-ways-to-read-a-text-file-line-by-line-in-python
3. Reading and writing files https://www.pythonforbeginners.com/files/reading-and-writing-files-in-python
4. Python Files I/O https://www.tutorialspoint.com/python/python_files_io.htm
5. Working with files in Python https://realpython.com/working-with-files-in-python/
6. File handling in Python https://www.geeksforgeeks.org/file-handling-python/
7. File operations in Python https://www.programiz.com/python-programming/file-operation

---

1. How to read a text file from a URL in Python https://www.kite.com/python/answers/how-to-read-a-text-file-from-a-url-in-python
2. Downloading files from web using Python https://www.tutorialspoint.com/downloading-files-from-web-using-python
3. An Efficient Way to Read Data from the Web Directly into Python without having to download it to your hard drive https://towardsdatascience.com/an-efficient-way-to-read-data-from-the-web-directly-into-python-a526a0b4f4cb

---

1. Web Requests with Python (using http and/or https) https://www.pluralsight.com/guides/web-scraping-with-request-python
2. Troubleshooting certificate errors (really common with https requests) https://stackoverflow.com/questions/27835619/urllib-and-ssl-certificate-verify-failed-error

In [ ]:

In [ ]: