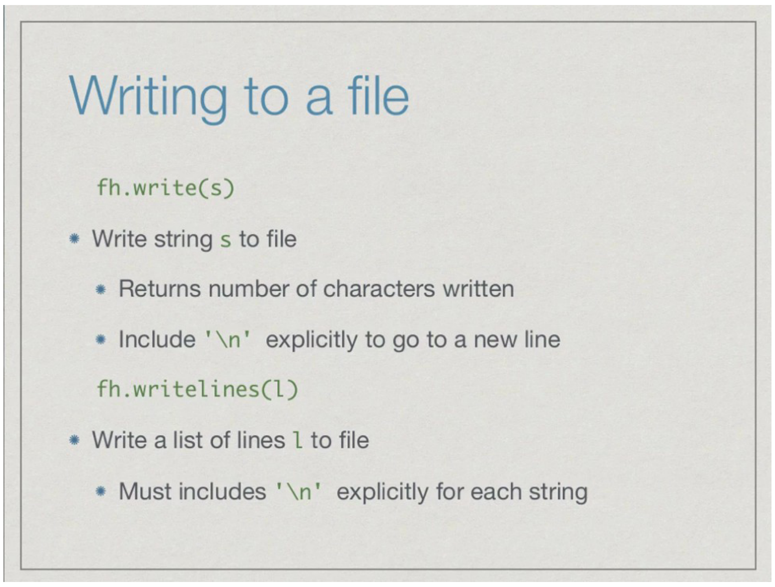
(Refer Slide Time: 10:44)



Writing to a file

fh.write(s)

- Write string s to file
  - Returns number of characters written
  - Include '\n' explicitly to go to a new line

fh.writelines(l)

- Write a list of lines l to file
  - Must includes '\n' explicitly for each string

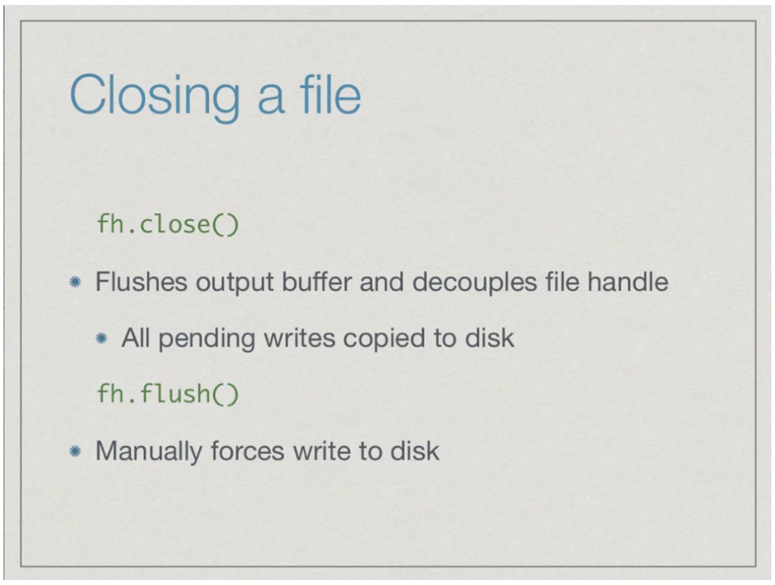
Having read from a file then the other thing that we would like to do is to write to a file. So, here is how you write to a file just like you have read a command you have a write command, but now unlike read which implicitly takes something from the file and gives to you, here you have to provide it something to put in the file. So, write takes an argument which is a string. When you say, write s says take the string s and write it to a file.

Now, there are two things; one is this s may or may not have a backslash n, it may have more than one backslash n. So, is nothing tells you this is one line part of a line more than a line you have to write s according to the way you want it to be written on the file, if you want it to be in one line you should make sure it ends with a backslash n.

And this write actually returns the number of characters written. Now, this may seem like a strange thing to do, why should it tell you because you know from the length of s what is number of character is written, but this is useful if, for instance, the disk is full. If you try to write a long string and find out only part of the string was written and this is a indication that there was a problem with the write. So, it is useful sometimes to know how many characters actually got written out of the characters that tried to write.

The other thing which writes in bulk to a file is called writelines. So, this takes list of strings and writes them one by one into the file. Now though it says write lines these may not actually be lines. So, its bit misleading the name if you want to them in lines you must make sure that you have each of them terminated by backslash n. If they are not then they will just cascade to form a long line thing. So, though it says writelines it should be more like write a list of strings, that should, that is a more appropriate name for this function, it just takes a list of strings and writes it to the file pointed to by the file handle.

(Refer Slide Time: 12:40)



Closing a file

```
fh.close()
```

- Flushes output buffer and decouples file handle
  - All pending writes copied to disk

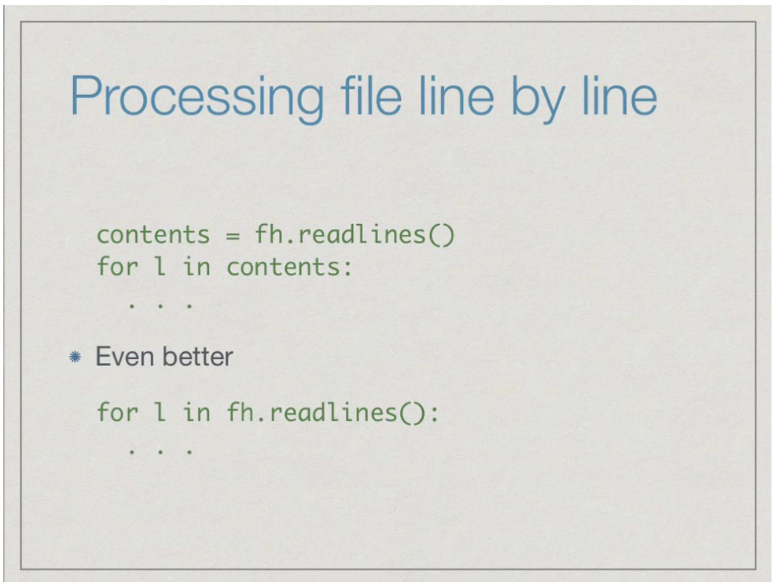
```
fh.flush()
```

- Manually forces write to disk

And finally, as we said once we are done with a file, we have to close it and make sure the buffers that are associated with the file, especially if you are writing to a file that they are flushed. So, fh dot close, will close the file handle fh and all pending writes at this point are copied out to the disk. It also now means that fh is no longer associated with the file we are dealing with. So, after this if we try to invoke operation on fh it is like having undefined name in python.

Now, sometimes there are situations where we might want to flush the buffer without closing the file. We might just want to make sure that all writes up to this point have been actually reflected on the disk. So, there is a command flush which does this. In case we say flush, it just say if there are any pending writes then please put them all on to the disk, do not wait for the risk drives to accumulate until the buffer is full and then write as you normally would to the disk.

(Refer Slide Time: 13:42)



Processing file line by line

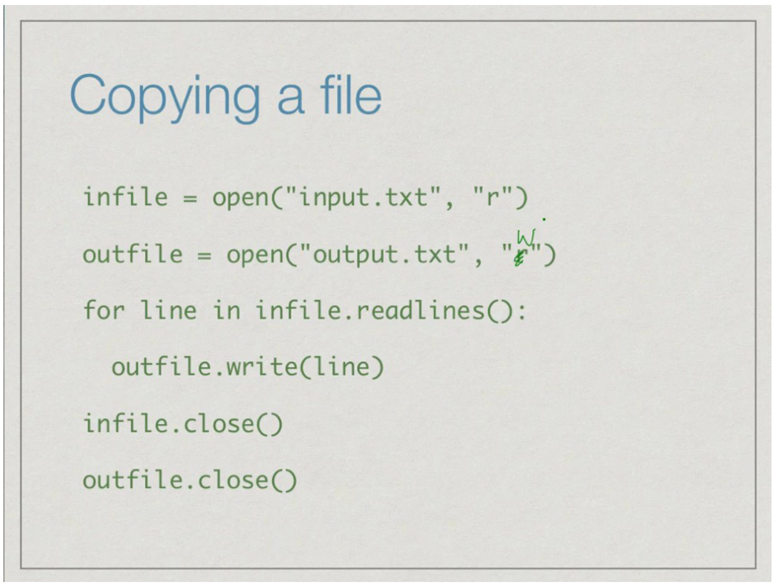
```
contents = fh.readlines()
for l in contents:
  . . .
```

- Even better

```
for l in fh.readlines():
  . . .
```

Here is a typical thing that you would like to do in python, which is to process it line by line. The natural way to do this is to read the lines into a list and then process the list using for. So, you say content is fh dot readlines and then for each line and contents you do something with it. You can actually do this in a more compact way, you can get do away with the name contents and just read directly every line return by the function fh dot readlines. So, this is the equivalent formulation of the same loop.

(Refer Slide Time: 14:25)



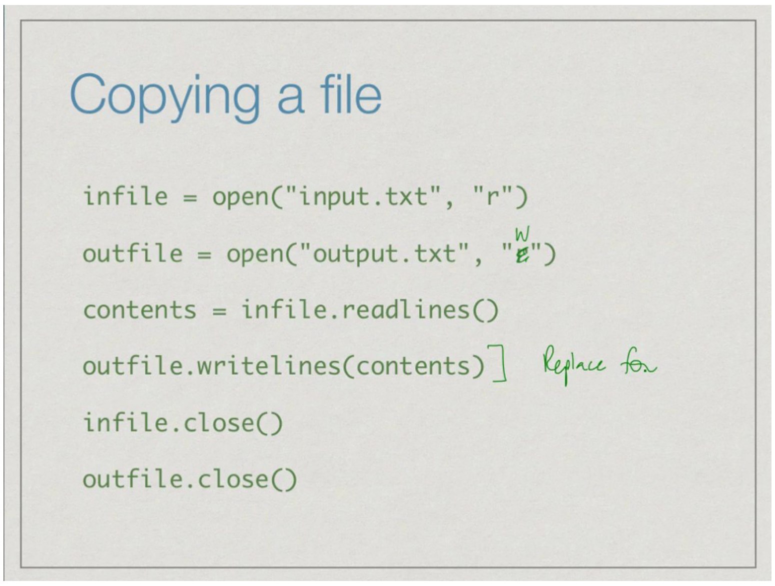
Copying a file

```
infile = open("input.txt", "r")
outfile = open("output.txt", "w")
for line in infile.readlines():
  outfile.write(line)
infile.close()
outfile.close()
```

As an example, for how to use this line by line processing, let us imagine that we want to copy the contents of a file input dot txt to a file output dot txt.

So, the first thing we need to do is to make sure that we open it correctly. We should actually open outfile with mode ‘w’ and in file mode ‘r’. This tells that I am going to read from infile and write to outfile. Now, for each line in returned by readlines on infile, remember that when I get line from readline the backslash n is already there, if I do not do anything to the backslash n, I can write it out the exactly the same way. So, for each line that I read from the list infile dot readlines I just write it to outfile and finally, I close both the files. This is one way to copy one file from input to output.

(Refer Slide Time: 15:25)

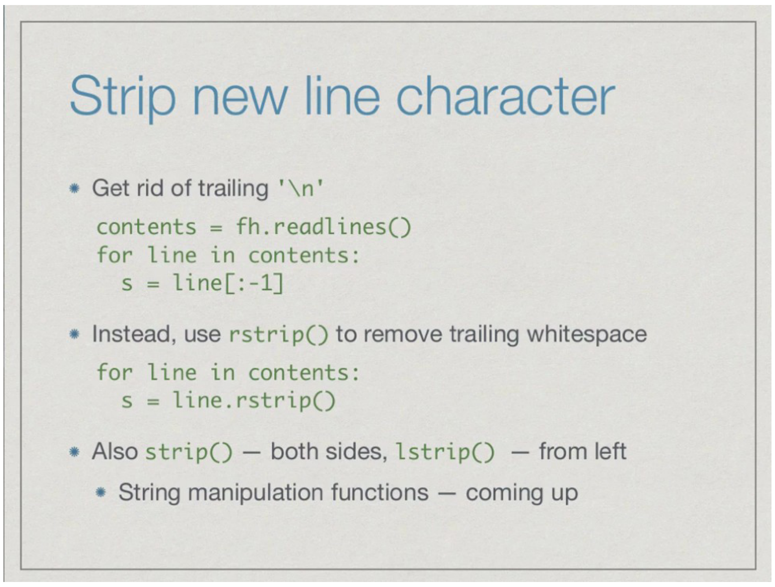


Copying a file

```
infile = open("input.txt", "r")
outfile = open("output.txt", "~~r~~w")
contents = infile.readlines()
outfile.writelines(contents)]  Replace for
infile.close()
outfile.close()
```

Of course, we can do it even in one shot because there is a command called writes lines, which takes the list of strings and writes them in one shot. So, instead of going line by line through the list readlines we can take the entire list contents and just output it directly through writelines, this is an alternative way where I have replaced. This is basically replacing the for. So, instead of saying for each line in infiles I can just write it directly out.

(Refer Slide Time: 16:02)



Strip new line character

- Get rid of trailing '\n'

```
contents = fh.readlines()
for line in contents:
  s = line[:-1]
```

- Instead, use rstrip() to remove trailing whitespace

```
for line in contents:
  s = line.rstrip()
```

- Also strip() — both sides, lstrip() — from left
    - String manipulation functions — coming up

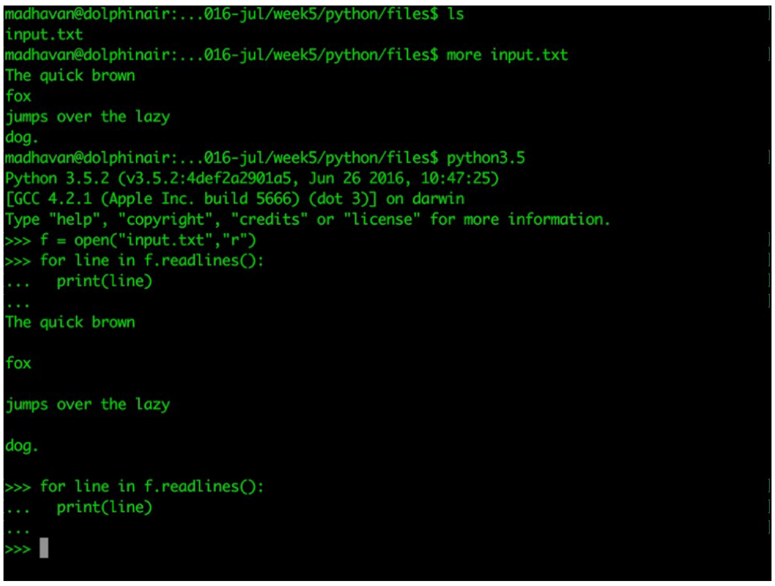
One of the things we are talking about is this new line character which is a bit of annoyance. If we want to get with a new line character, remember this is only a string and the new line character is going to be a last character in this string. So, one way to get is just to take slice of the string up to, but not including the last character. Now, remember that we when we count backwards minus 1 is the last character. If we will take the slice from 0 up to minus 1 then it will correctly exclude the last character from the string.

So, s is equal to line colon minus 1, will take the line and the strip of the last character which is typically backslash n that we get, when we do readlines. Now, in general we may have other spaces. So, remember when you write out text very often we cannot see the spaces the end of the line because they are invisible to us. These are what are called white space. So spaces, tabs, new lines, these are characters which do not display on the screen, especially spaces and tabs and there at a end of line, we do not know the line ends with the last character we see there are spaces after words.

So, r strip is a string command which actually takes a string and removes the trailing white space, all the white spaces are at end of the line. In particular there is only a backslash n and it will strip to a backslash n. It also strips to other jump there is some spaces and tabs before the backslash n and return that. So, s equal to line dot r strip that is strip line from the right of white space. This is an equivalent thing to the previous line except it is more general because strips all the white space not just by the last backslash n, but all the white spaces end of the line.

We can also strip from the left using l strip or we can strip on both sides if we just say strip without any characterization l or r. These are the string manipulation functions and we will look at some more of them, but this is just useful one which has come up immediately in the context of file handling. So, before we go ahead let us try and look at some examples of all these things that we have seen so far.

(Refer Slide Time: 18:05)



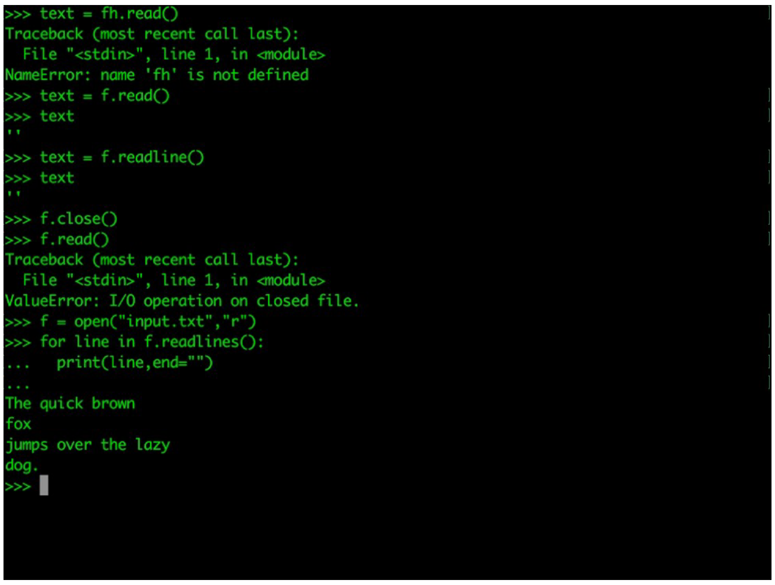
```
madhavan@dolphinair:...016-jul/week5/python/files$ ls
input.txt
madhavan@dolphinair:...016-jul/week5/python/files$ more input.txt
The quick brown
fox
jumps over the lazy
dog.
madhavan@dolphinair:...016-jul/week5/python/files$ python3.5
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> f = open("input.txt","r")
>>> for line in f.readlines():
...   print(line)
...
The quick brown

fox

jumps over the lazy

dog.

>>> for line in f.readlines():
...   print(line)
...
>>> 
```

We have created a file called input dot txt which consist of a line, the quick brown fox jumps over the lazy dog. Now, let us open the python interpreter and try to read lines from this file and print it out. So, we can say, for instance, that f is equal to open input dot txt in read. Now, I have opened the file and now I can say, for instance, for line in f dot readlines print line. Now, you will see something interesting happening here, you will see that we have now a blank line between every line our file.

Now, why is there blank lines between every line in our file that is because when we readlines we get a backslash n character from the line itself. So, the quick brown, the first line end with the backslash n, fox end with backslash n and then over and above that if you remember the print statement adds a backslash n of its own. So, actually print is putting out to blank lines for each of these.

Now, let us try and do this again, supposing I repeat this thing and now I do this again, now nothing happens the reason nothing happens is because we had this sequential reading of the file. So, the first time we did f dot readlines, it read one line at a time and now we are actually pointing to the end of the file.

(Refer Slide Time: 19:49)

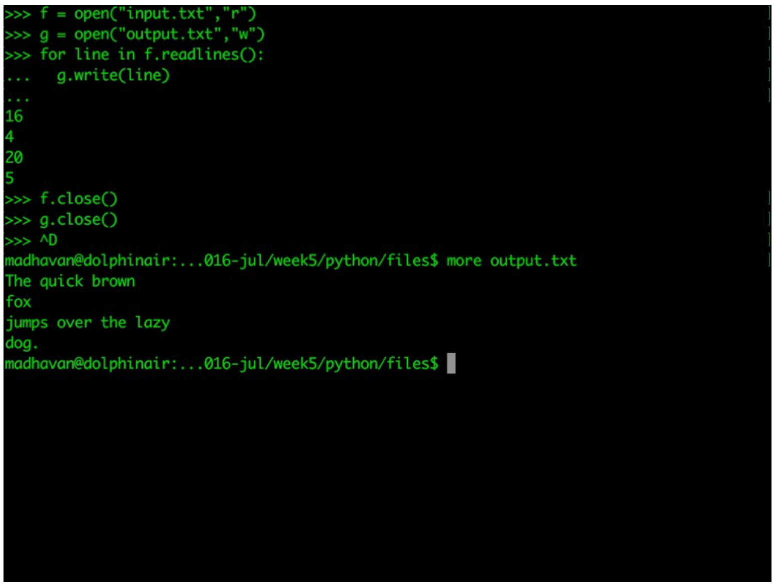


```
>>> text = fh.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'fh' is not defined
>>> text = f.read()
>>> text
''
>>> text = f.readline()
>>> text
''
>>> f.close()
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
>>> f = open("input.txt","r")
>>> for line in f.readlines():
...   print(line,end="")
...
The quick brown
fox
jumps over the lazy
dog.
>>> 
```

If for instance, at this point we were to say text equal to fh dot read, sorry f dot read, then text will be empty string. This is the indication that we have actually reached the end of the file. Similarly, if we try to say readline again text will be empty string. So, remember we said that if read or readlines returns the empty string then we have reached the end of the file. The only way we can undo this is to start again by closing the files. So, what we say is f dot close. This closes of the file.

Now, if you try to do f dot read then we will get an error saying that this is not being defined. So, we do not have f with us anymore. So, again we have to say f is open input dot txt r and now we can say while for line in f dot readlines for each line. Supposing, we now use that trick that we had last time which is to say end equal to empty string that says do not insert anything after each print statement. Now, if you do this you see we get back to exactly the input files as it is without the extra blank lines because print is no longer creating these extra lines.

(Refer Slide Time: 21:10)

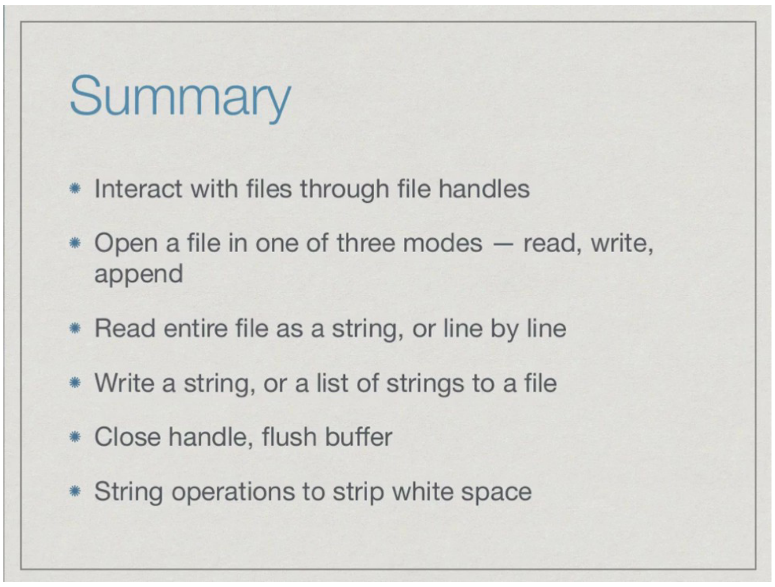


```
>>> f = open("input.txt","r")
>>> g = open("output.txt","w")
>>> for line in f.readlines():
...   g.write(line)
...
16
4
20
5
>>> f.close()
>>> g.close()
>>> ^D
madhavan@dolphinair:...016-jul/week5/python/files$ more output.txt
The quick brown
fox
jumps over the lazy
dog.
madhavan@dolphinair:...016-jul/week5/python/files$
```

Let us say we want to copy input dot txt to a output dot txt, we say f is equal to open input dot txt r as before and we say g is equal to open output dot txt w and now we say for line in f dot readlines, g dot write line.

Now, notice you get the sequence of numbers why do we get a sequence of numbers that is because each time we write something it returns a number of character written and it will turn out, if you look at the lines quick brown fox, etcetera, for example, the second line is just fox, fox has three letters, but if you include the backslash n its wrote 4 letters. That is why quick brown was 15 letters plus a backslash n, fox was a 3 plus backslash n. So, this is line by line. Now, if I correctly close these files then come out of this, then output dot txt is exactly the same as input dot txt.

(Refer Slide Time: 22:26)


Summary

* Interact with files through file handles
* Open a file in one of three modes — read, write, append
* Read entire file as a string, or line by line
* Write a string, or a list of strings to a file
* Close handle, flush buffer
* String operations to strip white space

To summarize what we have seen is that, if you want to interact with files we do it through file handles, which actually corresponds to the buffers that we use to interact between the memory and the file on the disk. We can open a file in one of three modes; read, write and append. We did not actually do an example with the append, but we do append what it will do, keep writing beyond where the file already existed, otherwise write will erase the file and start from the beginning.

We saw that read, readline and readlines, using this we can read the entire file in one shot of the string or read it line by line. Similarly, we can either write a string or we write a list of strings too. So, we have a write command in a writelines and writelines are more correctly to be interpreters write list of strings.

Finally, we can close the handle when we have done and in between that we can flush the buffer by using flush command and we also saw that there are some string operations to strip white space and this can be useful to remove these trailing backslash n which come whenever you are processing text files.