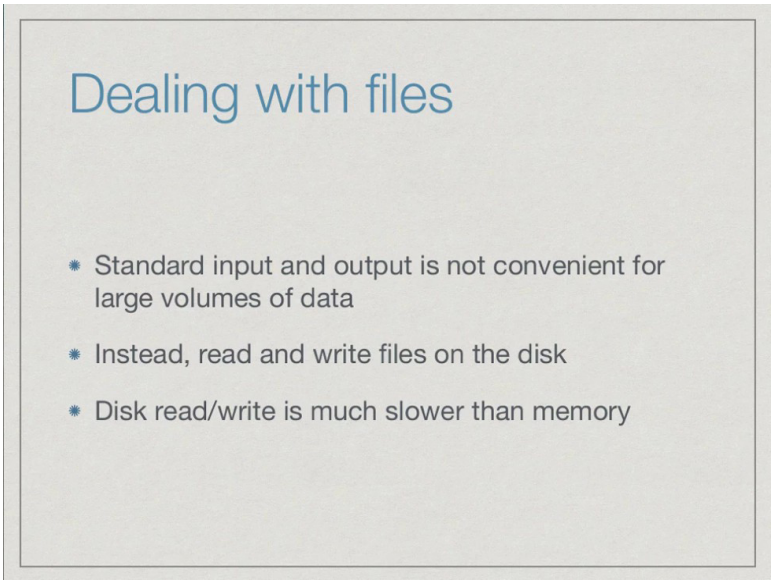
Programming, Data Structures and Algorithms in Python
Prof. Madhavan Mukund
Department of Computer Science and Engineering
Chennai Mathematical Institute,Madras

Week - 05
Lecture - 03
Handling files

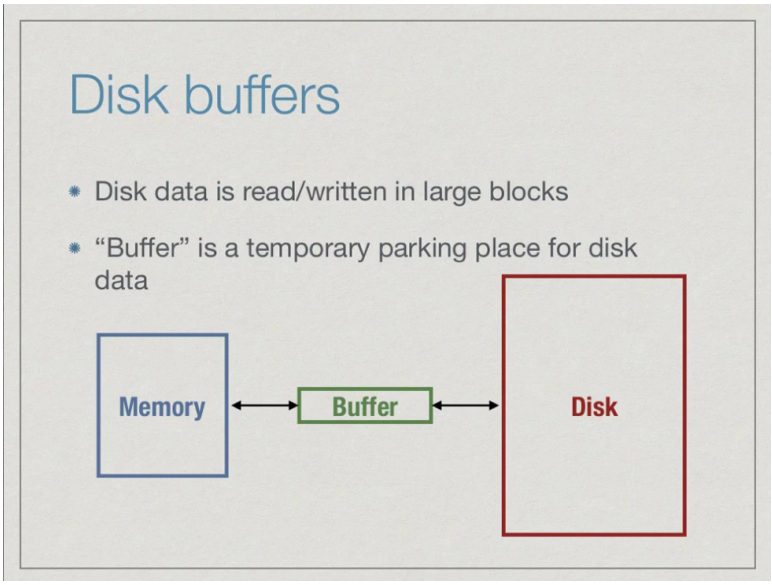
(Refer Slide Time: 00:02)



In the last lecture we saw how to use the input and print statements to collect input from the standard input that is the keyboard, and to display values on the screen using print.

Now, this is useful for small quantities of data, but we want to read and write large quantities of data. It is impractical to type them by hand or to see them as a scroll pass in this screen. So, for large data we are forced to deal with files which reside on the disk. So, we have to read a large volume of data which is already written on a file in the disk and the output we compute is typically return back into another file on the disk. Now, one thing to keep in mind when dealing with disks is that disk read and write is very much slower than memory read and write.

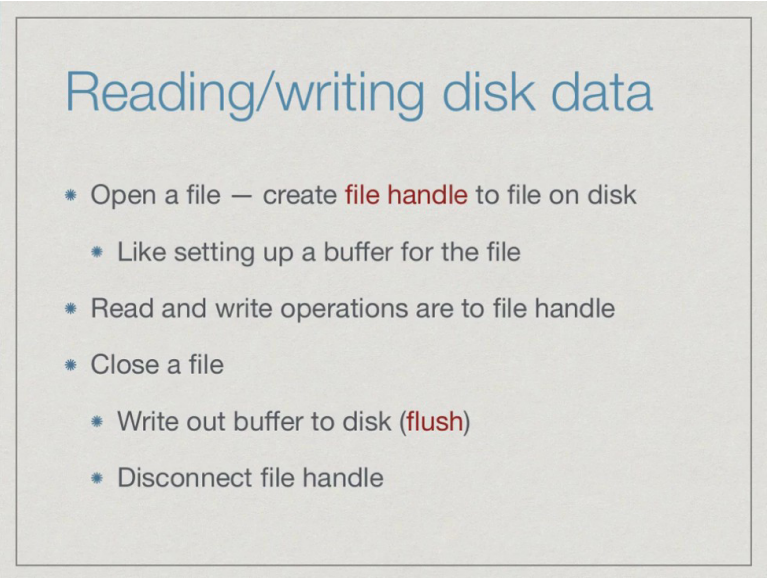
(Refer Slide Time: 00:48)



To get around this most systems will read and write data in large blocks. Imagine that you have a large storage facility in which you store things in big cartons. Now, when you go and fetch something you bring a carton at a time even if you are only looking for, say one book in that carton, you do not go and fetch one book out of the carton from the storage facility, you bring the whole carton and then when you want put things back again you assemble them in a carton and put them back.

In the same way, the way that data flows back and forth between memory and disk is in chunks called blocks. So, even if you want to read only one value or only one line it will actually a fetch large volume of data from the disk and store it in what is called a buffer and then you read whatever you need from the buffer. Similarly, when you want to write to the disk you assemble your data in the buffer when the buffer is enough quantity to be written on the disk then one chunk of data or block is written back on the disk.

(Refer Slide Time: 01:49)

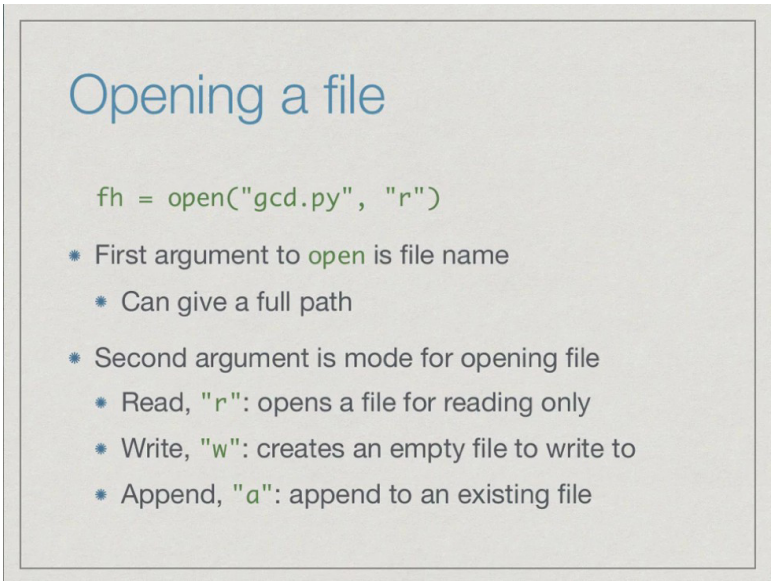


When we read and write from a disk the first thing we need to do is connect to this buffer. This is called opening a file. So, when we open a file we create something called a file handle and you can imagine that this is like getting access to a buffer from which data from that file can read into memory or written back.

Now, having opened this file handle everything we do with the file is actually done with respect to this file handle. So, we do not directly try to read and write from the disk, instead we read and write from the buffer that we have opened using this file handle and finally, when we are done with our processing we need to make sure that all the data that we have written goes back. So, this is done by closing the file.

So, closing the file has two effects; the first effect is to make sure that all changes that we intended to make to the file. Any data we want to write to the file is actually taken out to the buffer and put on to the disk and this technically called flushing the buffer. So, closing a file flushes the output buffer make sure that all rights go back to the file and do not get lost and the second thing it does is that it in some sense makes this buffer go away. So, it disconnects the file handle that we just set up. Now, this file is no longer connected to us if we want to read or write it again we have to again open it.

(Refer Slide Time: 03:11)



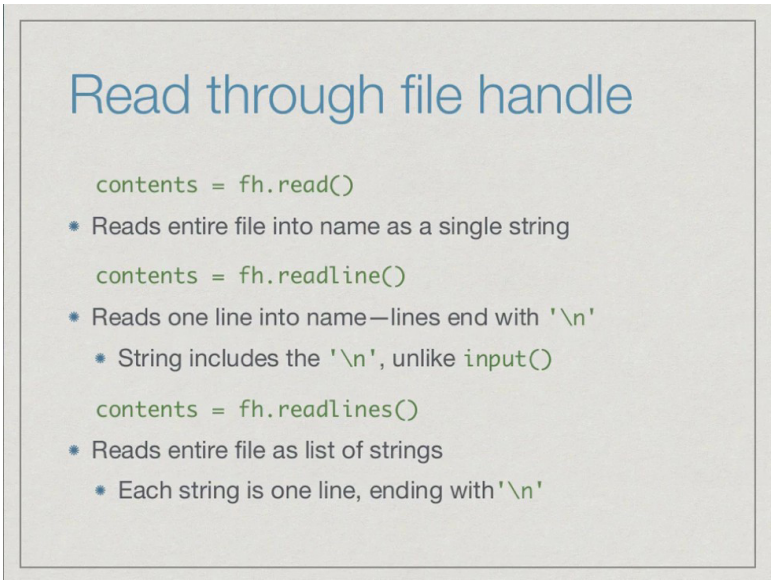
The command to open a file is just ‘open’. The first argument that you give open is the actual file name on your disk. Now, this will depend a little bit on what system you are using, but usually it has a first part and an extension. This commands, for instance, to open the file gcd dot py. Now implicitly, if you just give a file name it will look for it in the current folder or directory where you running the script. So, you can give a file name which belongs to the different part of your directory hierarchy by giving a path and how you describe the path will depend on whether you are working on Windows or Unix, what operating system you are using.

Now, you see there is a second argument there, which is letter ‘r’. This tells us how we want to open the file. So, you can imagine that if you are making changes to a file by both reading it and writing it, this can create confusion. So, what we have to do is decide in advance whether we are going to read from a file or write to it, we cannot do both. There is no way we can simultaneously read from a file and modify it while it is open. So, read is signified by ‘r’.

Now, write comes in two flavors, we might want to create a new file from scratch. In this case we use a letter ‘w’. So, ‘w’ stands for write out a new file, we have to be bit careful about this because if we write out a file which already exits then opening it with more ‘w’ will just overwrite the contents that we already had. The other thing which might be useful to do is to take a file that already exists and add something to it. This is called

append. So, we have two writing modes; ‘w’ for write and ‘a’ for append. What append will do is it will take a file which already exists and add the new stuff the writing at the end of the file.

(Refer Slide Time: 05:02)



Once we have a file open, let us see how to read. So, we invoke the read command through the file handle. This is like some of the other function that you saw with strings and so on, where we attach the function to the object. So, fh is the file handle we opened, we want to read from it. So, we say fh dot read, what fh dot read does is it swallows the entire contents the file as a single string and returns it and then we can assign it to any name, here we use the name contents. So, contents is now assigned the entire data which is in the file handle pointed by fh in one string.

Now, we can also consume a file, we are typically dealing with text files. So, text file usually consists of lines; think of python code, for example, we have lines after lines after lines. A natural unit is a bunch of texts which is ended with new line character. If you remember this is what the input command does, the input command waits for you type something and then you press return which is a new line and whatever you type up to the return is then transmitted by input as a string to the name that you assigned to the input.

So, readline is like that, but the difference between the readline and input is that, when you read a line you get the last new line character along with the input string. When you

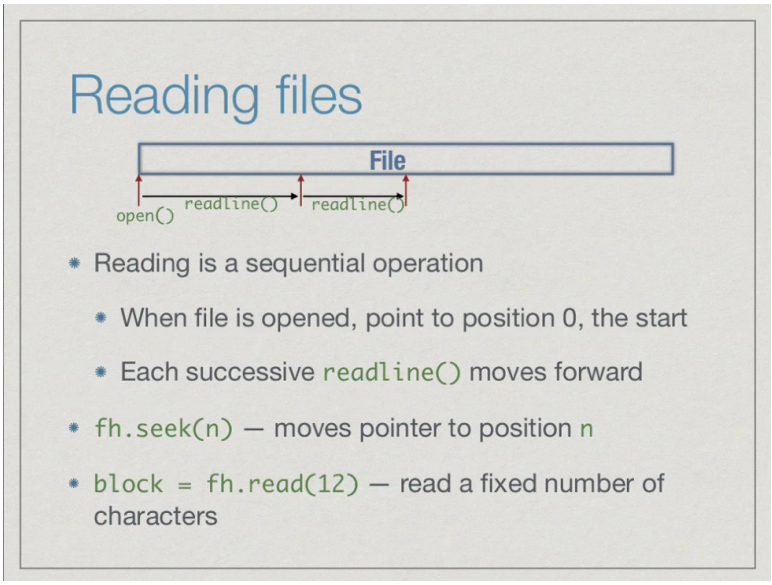
say input you only get the characters which come before the last new line the new line is not included, but in readline you do get the new line character.

So, you have to remember that you have the extra character floating around at the end of your string. So, this is conventionally the noted backslash n. The backslash n is a notation which denotes a single character even though looks two characters. This is supposed to be the new line character. Now, the actual new line character differs on operating systems from one to the other, but in python if we use backslash n and it will correctly translated in all the systems that you are using.

The third way that you can read from a file is to read all the lines one by one into a list of strings. So, instead of readline, if I say readlines then it reads the entire the files as a list of srings. Each string is one item in the list and remember again each of these lines has the backslash n included. So, read, readline and readlines, none of them will actually remove the backslash n. They will remain faithfully as part of your input.

In other words, if you are going to transfer this from one file to another, you do not want to worry reinserting the backslash n because this is already there. So, you can use this input output directly, but on the other hand, if you want to do some manipulation of the string then you must remember this backslash n is there and you must deal with it appropriately.

(Refer Slide Time: 07:49)



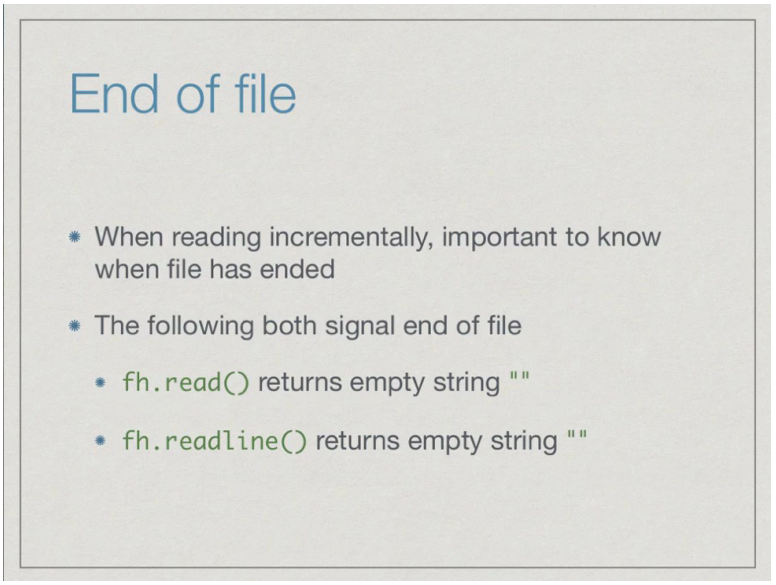
Reading files is inherently a sequential operation. Now, of course, if we use the basic command read, it reads the entire content. So obviously, it reads from beginning to the end, but if you are reading one line at a time then the way it works is that when we open the file we are initially at the beginning of the file. So, you can imagine a pointer like this red arrow which tells us where we are going to read next. So, initially when we open we are going to read from the beginning, now each readline takes us forward. If I do a readline at this point it will take me up to the next backslash n.

Remember a line is a quantity which is delimited by backslash n. So, we could have a line which has 100 characters, next line could have 3 characters and so on. It is from one backslash n to the next is what a line, so this is not a fixed link. So, we will move forward reading one character at a time until we have backslash n, then everything up to the backslash n will be returned as the effect to a string return by the readline and pointer move to the next character. Now, we do another readline possibly of different line again the point to move forward. So, in this way we go from beginning to the end.

In case we want to actually divert from the strategy there is a command seek, which takes a position, an integer n, and moves directly to the position n regardless of where you are. This is one way to move back or to jump around in a file other than by reading consecutively line by line.

Finally, we can modify the read statement to not to read the entire file, but to read a fix number of characters. Now, this may be useful if your character actually your file actually consists of fix blocks of data. So, you might have say, for example, pan numbers which are typically 10 characters long and you might have just stored them as one long sequence of text without any new lines knowing that every pan number is 10 characters. So, if we say fh dot read 10, it will read the next pan number and keep going and this will save you some space in the long run. So, there are situation where you might exploit this where you read a fix number of characters.

(Refer Slide Time: 09:56)



When we are reading a file incrementally, it is useful to know when the file is over because we may not know in advance how long files is or how many lines of file is. So, if you are reading a file line by line then we may want to know when the file has ended. So, there are two situations where we will know this. So, one is if we try to read using the read command and we get nothing back, we get an empty string that means the file is over, we have reached end of file.

Similarly, if we try to read a line and we get empty string it means we reached the end of file. So, read or readline if they return empty string its means that we have reached the end of the file. Remember, we are going sequential from beginning to the end. So, we reached the end of the file and there is nothing further to read in this file.