Text Files:

1) Opening and Writing:

```
f=file.open(“myfile.txt”,’w’)
```

This line creates a file if it does not exist. If it does exist, then the full file is deleted, and a new one is created in its place. Never forget that the mode is also a string. You’ll cause a bunch of errors if you’re not careful.

When in the write mode, only writing can be done.

```
f.write(“Blah Blah Blah”)
```

The write method takes only a string argument. No other data type is ok. For whatever reason, the write function returns the number of characters it writes into the file. I have no idea how this can be useful.

```
f.close()
```

This closes the file, and commits the changes made into the file. If you open the file in write mode, write a bunch of stuff and do not close it, the changes you made will NOT reflect on the actual text file. (However, the moment you open an existing file in write mode, the existing data is wiped clean.)

To add more data to the end of an existing text file, you must open it in the append mode. This mode also allows only for writing to be done.

```
f=file.open(“myfile.txt”,’a’)
```

The pointer automatically goes to the end of the file, so any data you write will automatically go to the end of the file.

```
f=file.open(“myfile.txt”,’r’)
```

The read mode is exclusively for reading data. More on the specific commands on this in a bit.

```
f=file.open(“myfile.txt”,’w+’)
```

Write + Read. If the file exists, it’s destroyed. You are permitted to read the file, and move around the pointer, so if you’re not careful about the pointer location, you’ll unwittingly overwrite some text.

```
f=file.open("myfile.txt",'a+')
```
Append + Read. The existing file will NOT be destroyed. The file pointer starts at the end, and you're permitted to do whatever you want.

```
f=file.open("myfile.txt",'r+')
```
Read + Write. This is extremely similar to a+ mode, because this won't delete any existing data either, and you have total freedom to do whatever you want. The only difference is that the pointer starts at the beginning of the file.

- Step 1: There are 6 modes of opening the file, Only w, and w+ create a fresh file, and the rest just throw an error if the file doesn't exist. A is useful if you want to just add new data and not screw anything over, Same story for R, limited power to not screw anything up. R+ and A+ are total freedom, but A+ is more convenient if you're going to add new data right away, and don't want to mess with the pointer.

- Step 2: Write to a file with proper use of "\n" to space out things in new lines. This is something you'll need to look out for when you read the file.

- Step 3: Read, Go Forward, Go Back, Go Crazy. Knock Yourself Out.

2) <u>Reading:</u>

```
f.read(x)
```
This will read exactly x characters starting from the present location of the pointer. In the absence of an argument, it reads the entire file. It returns a single string containing whatever it read.

If you ask it to read when it's already at the end of the file, it will just give you Null String.

"/n" is ONE character. Conventionally, we do NOT put a newline at the end of the file. Since the file is over, this is considered redundant, but it can be a pain if you're appending data.

```
f.tell()
```
Returns the pointer location.

```
f.seek(x)
```

This takes the pointer to the x^{th} byte in the text file. Position 0 means read(1) returns character 0. Position x means you're just about to read the character at position x. There seems to be this one really irritating thing here: "\n" is considered to be 2 bytes by the pointer. This, of course, is totally sensible, but because read/write and seek don't use the same counting system, it drives me ever so slightly crazy.

So, if f is "BLAHBLAH\nBLAHBLAH"

Then

```
f.seek(0)
print(f.read(4))
print(f.tell())
```

gives

```
BLAH
4
```

Wheras

```
f.seek(0)
print(f.read(9))
print(f.tell())
```

gives

```
BLAHBLAH
10
```

Also note that once the file is fully read the pointer points just after the last character, and that's why reading further gives the null string.

Concretely, if f is "BLAHBLAH"

```
f.seek(0)
print(f.read())
print(f.tell())
```

gives

```
BLAHBLAH
8
```

Now, onto the other reading functions.

```
f.readline()
```

This reads exactly one line and returns it to you.

The weird part is, this actually takes an argument.

```
f.readline(x)
```

You'd imagine it means that it will read x lines, but Nope! It works EXACTLY like a read function, read x CHARACTERS from the pointer location, until it hits a endline, and which point it just stops reading. Note that it will read till endline or x characters, whichever comes earlier.

Look, If f is "BLAHBLAH\nBLAHBLAH\nFOURFOUR"

```
f.seek(0)
print(f.readline(4))
```

gives

```
BLAH
```

Wheras

```
f.seek(0)
print(f.readline(20))
```

gives

```
BLAHBLAH
```

The **default** argument that says "Read the whole line" is

```
f.readline(-1)
```

Okay! Thank you, Next!

```
f.readlines(x)
```

This returns a LIST of strings, each line in one string. Now, this also takes an argument, and guess what, it doesn't take in the number of lines to read either.

Instead, the argument tells IF the total number of bytes you have seen so far, are beyond this x, then DON'T add the next line to the list. STOP with just this line.

If f is “BLAHBLAH\nBLAHBLAH\nFOURFOUR”

```
f.seek(0)
print(f.readlines(4))
```

gives

```
[“BLAHBLAH\n”]
```

It read line1, the size is beyond 4 bytes, so it doesn’t read line 2.

Similarly,

```
f.seek(0)
print(f.readlines(10))
```

gives

```
[“BLAHBLAH\n”, “BLAHBLAH\n”]
```

The weird part is that this argument doesn’t seem to count newline characters as bytes at all. Here, line one has 8 bytes. 8 is not >=10 so proceed.

Demonstartion of the \n weirdness:

```
f.seek(0)
print(f.readlines(9))
```

gives

```
[“BLAHBLAH”, “BLAHBLAH”]
```

```
f.seek(0)
print(f.readlines(8))
```

```
[“BLAHBLAH\n”]
```

I expected 9 and 8 to give the same result , but they clearly don’t.

Moral of the story, just don’t use the arguments in readline or readlines unless you have some super-specific use for it and can afford mild headaches, chronic depression and occasional suicidal tendencies.

Some additional information about useful string methods:

```
s=”   blahblahblah\n”
s.strip()
```

strip deletes all the leading and trailing spaces and newlines. now, s is “blahblahblah”

```
s=”blah,blah,blah”
x=s.split(“,”)
```

Split creates a list of elements using the argument given as the separator. x is ['blah', 'blah', 'blah'] In the absence of any argument, the separator is taken to be a space.