File Handling

Reading from and writing to file

Files

- A *file* is a sequence of bytes stored on a secondary memory device
- Text File : text document, spreadsheet, html file
    - A text file contains a sequence of characters that are encoded using some encoding (ASCII, utf-8, etc)
- Binary file: sequences of bytes with no encoding.
    - Executable files, image or audio files

Opening and closing a file

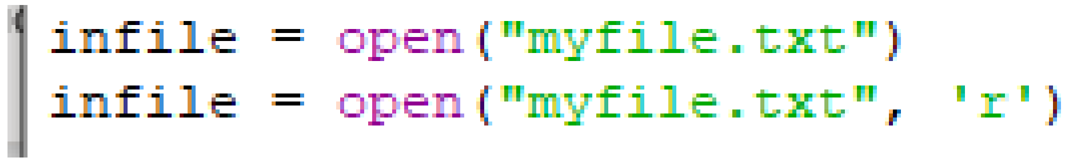
Steps to process a file:

1. *Opening* a file for reading or writing (`open()`)
2. *Reading* from the file and /or writing to the file (process the file)
3. *Closing* the file (`close()`)

open() function

- The function `open()` takes three arguments:
  - a file name,
  - a mode (optional)
  - an encoding (optional)

- To open a file ***myfile.txt*** use `open()`



```
infile = open("myfile.txt")
infile = open("myfile.txt", 'r')
```

- The file name is the *absolute* or *relative* pathname
- If no file exists an *exception* occurs

absolute and *relative* pathnames

- The ***absolute pathname*** of a file consists of the sequence of folders, starting from the root directory, that must be traversed to get to the file
- The ***absolute pathname*** is represented as a string in which the sequence of folders is separated by forward(/) or backward(\) slashes, depending on the operating system.

```
C:\2ndYearPython\PyCode\test.py
```

- The ***relative pathname*** of a file consists of the sequence of folders, starting from the *current working directory*, that must be traversed to get to the file

```
\PyCode\test.py
```

  - In UNIX Linux and Mac OS X systems the forward slash / is used as delimiter in a path. In Microsoft backslash \ is used
  - Python will however accept the forward slash / on a window system.

Mode (r, w, a, r+)

- The *mode* is a string that specifies how we interact with the opened file.
- The default is `r`

`r`	reading
`w`	writing
`a`	appending
`r+`	reading and writing

- Can also have `t` text or `b` binary

```
infile = open("myfile.txt", 'r')
```

- Text or ‘t’ is the default (if nether t or b specified)

Mode

Mode	Description
r	Reading mode (default)
w	Writing mode: if file already exists, its content is wiped. If not it is created
a	Append mode: writes are appended to the end of the file
r+	Reading and writing mode
t	Text mode (default)
b	Binary mode

- The difference in opening a file as text or binary is that binary files are treated as a sequence of bytes and are not decoded when read or encoded when written to
- Text files however, are treated as encoded files using some encoding

`open()`

- `open()` returns an *object* of an *Input* or *Output Stream* type that supports methods to *read* and/or *write* characters
  - This is a *file* object
- Different modes give us file objects of different file types

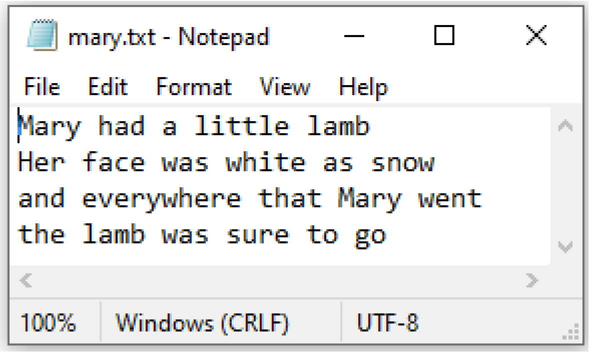
Some file methods

`infile.read()`	Read chars from `infile` until the end of the file (EOF) is reached. Return characters as a string
`infile.read(n)`	Read *n* chars from `infile` or until the end of the file is reached. Return characters as a string
`infile.readline()`	Read line by line from `infile`. Read until end of line character or until the end of the file is reached. Return characters as a string
`infile.readlines()`	Read lines from `infile` until the end of the file is reached. Returns s a `list` of lines.
`outfile.write(s)`	Write string s to `outfile`
`file.close()`	Close the file

`read` methods

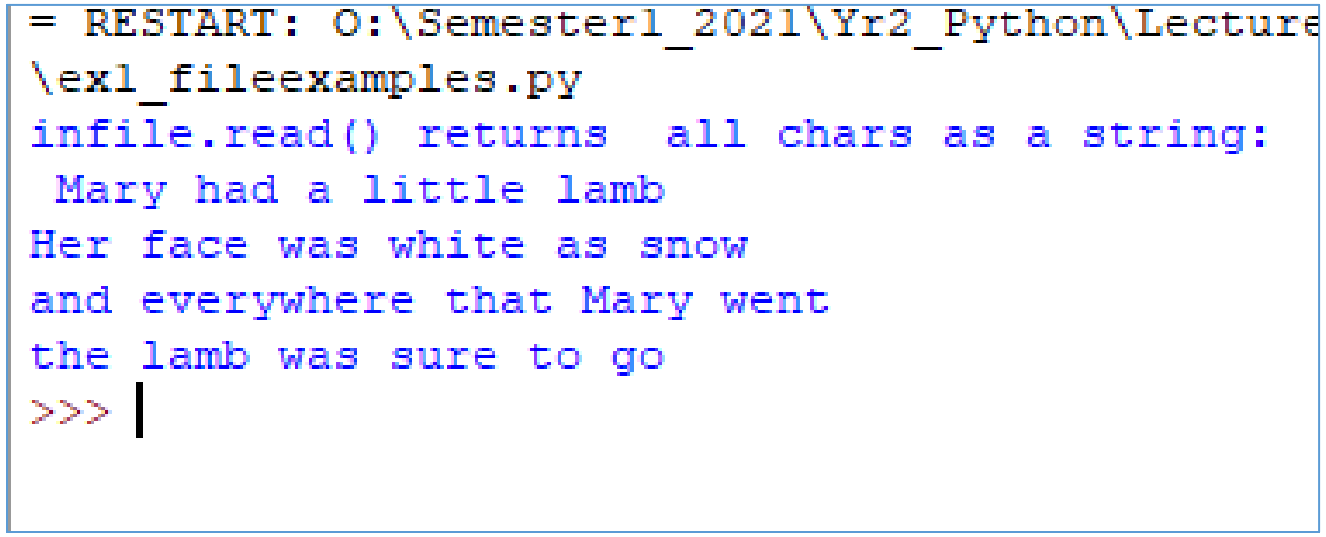
- The *read* methods are used to read the content of the file in different ways
- When a file is opened a cursor points to a character in the file
  - Usually the beginning of the file (when opened to read)
- When writing to a file the *writes* will be written starting at the cursor position

read()



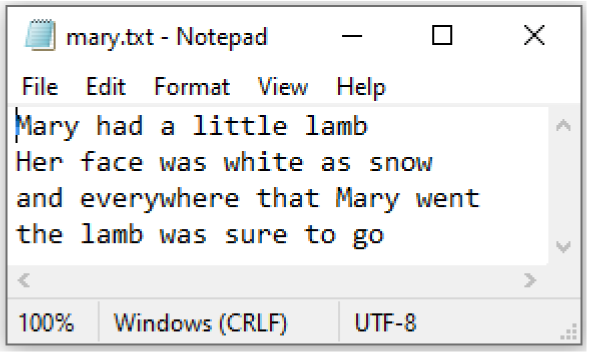
```
Mary had a little lamb
Her face was white as snow
and everywhere that Mary went
the lamb was sure to go
```

```python
## test read() methods
infile = open("mary.txt")
##read()  reads all chars  returns  a string
print("infile.read() returns  all chars as a string:\n" ,infile.read())
infile.close()
```



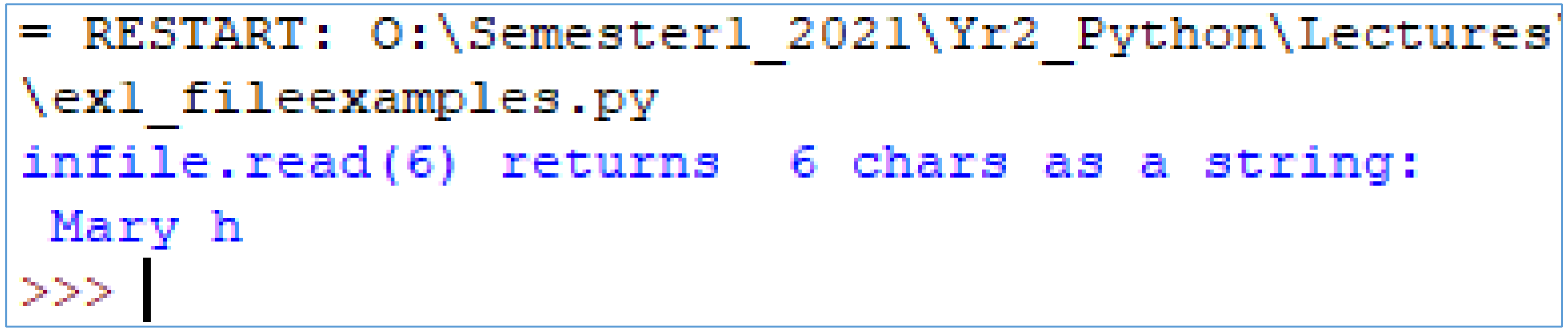
```
= RESTART: O:\Semester1_2021\Yr2_Python\Lecture
\ex1_fileexamples.py
infile.read() returns  all chars as a string:
 Mary had a little lamb
Her face was white as snow
and everywhere that Mary went
the lamb was sure to go
>>> 
```

read(n)



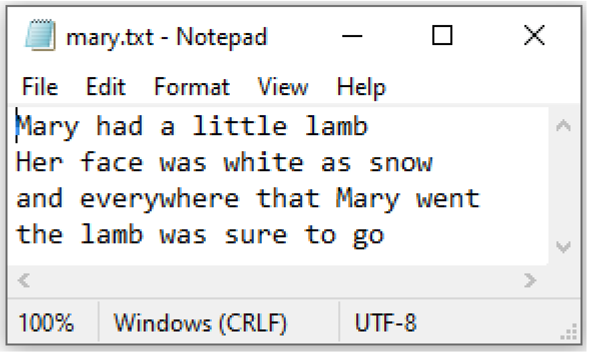
```
Mary had a little lamb
Her face was white as snow
and everywhere that Mary went
the lamb was sure to go
```

```python
infile = open("mary.txt")
##read(n)  reads n chars  returns  a string
print("infile.read(6) returns  6 chars as a string:\n" ,infile.read(6))
infile.close()
```

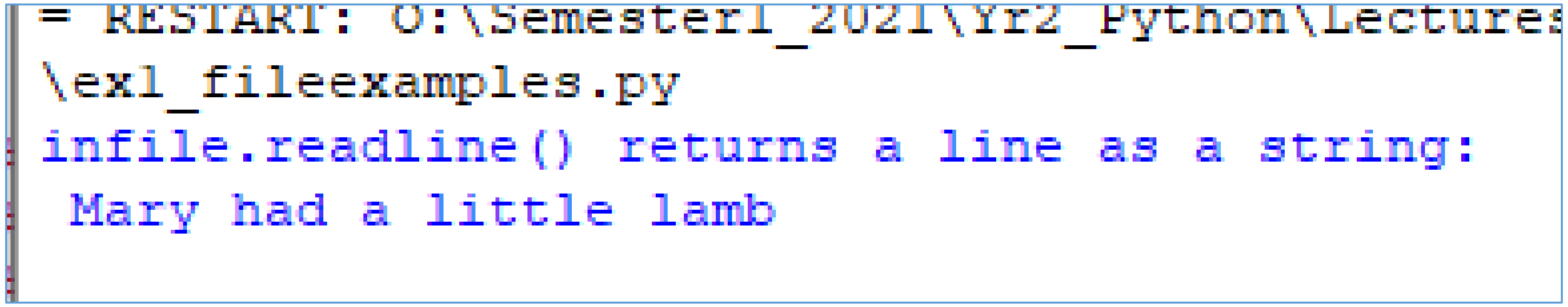


```
= RESTART: O:\Semester1_2021\Yr2_Python\Lectures
\ex1_fileexamples.py
infile.read(6) returns  6 chars as a string:
 Mary h
>>> 
```

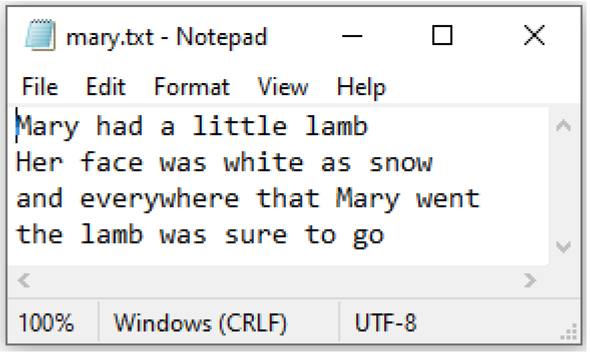
readline()


```
Mary had a little lamb
Her face was white as snow
and everywhere that Mary went
the lamb was sure to go
```

```python
infile = open("mary.txt")
##readline() reads chars up to end of current line  returns  a string
print("infile.readline() returns a line as a string:\n" ,infile.readline())
infile.close()
```

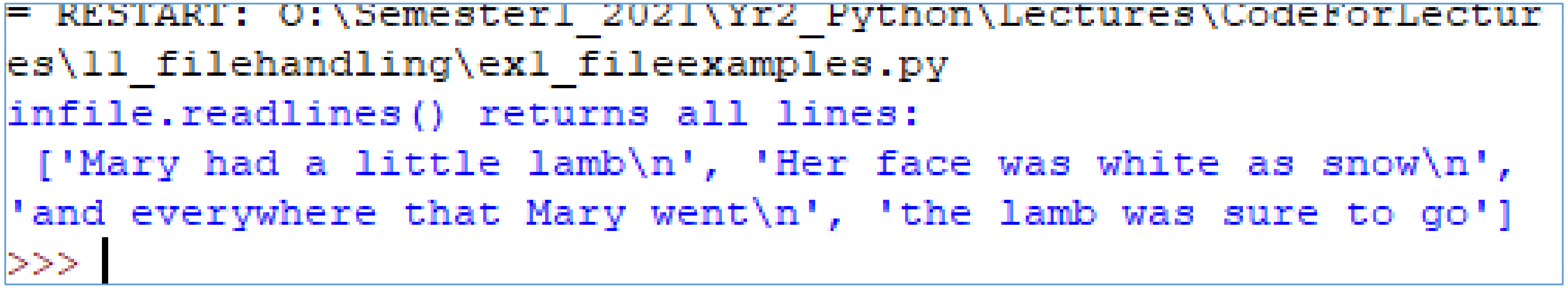

```
= RESTART: O:\Semester1_2021\Yr2_Python\Lectures
\ex1_fileexamples.py
infile.readline() returns a line as a string:
 Mary had a little lamb
```

readlines()

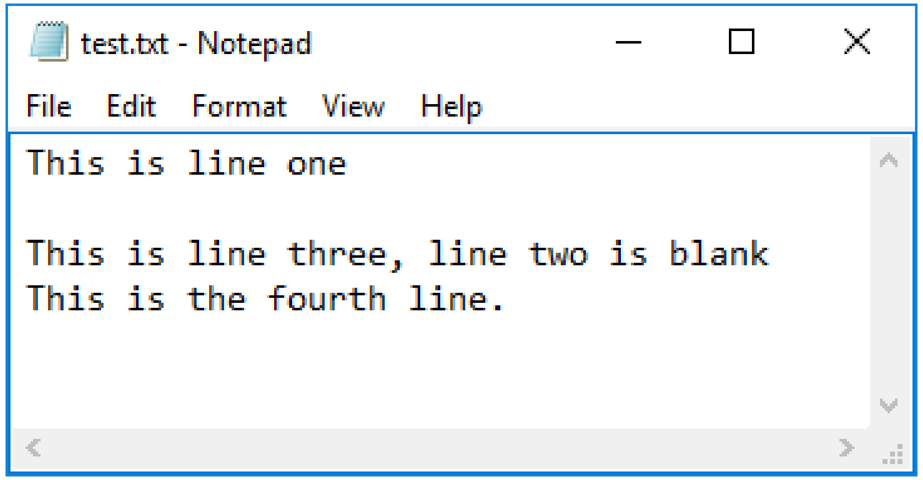


```
Mary had a little lamb
Her face was white as snow
and everywhere that Mary went
the lamb was sure to go
```

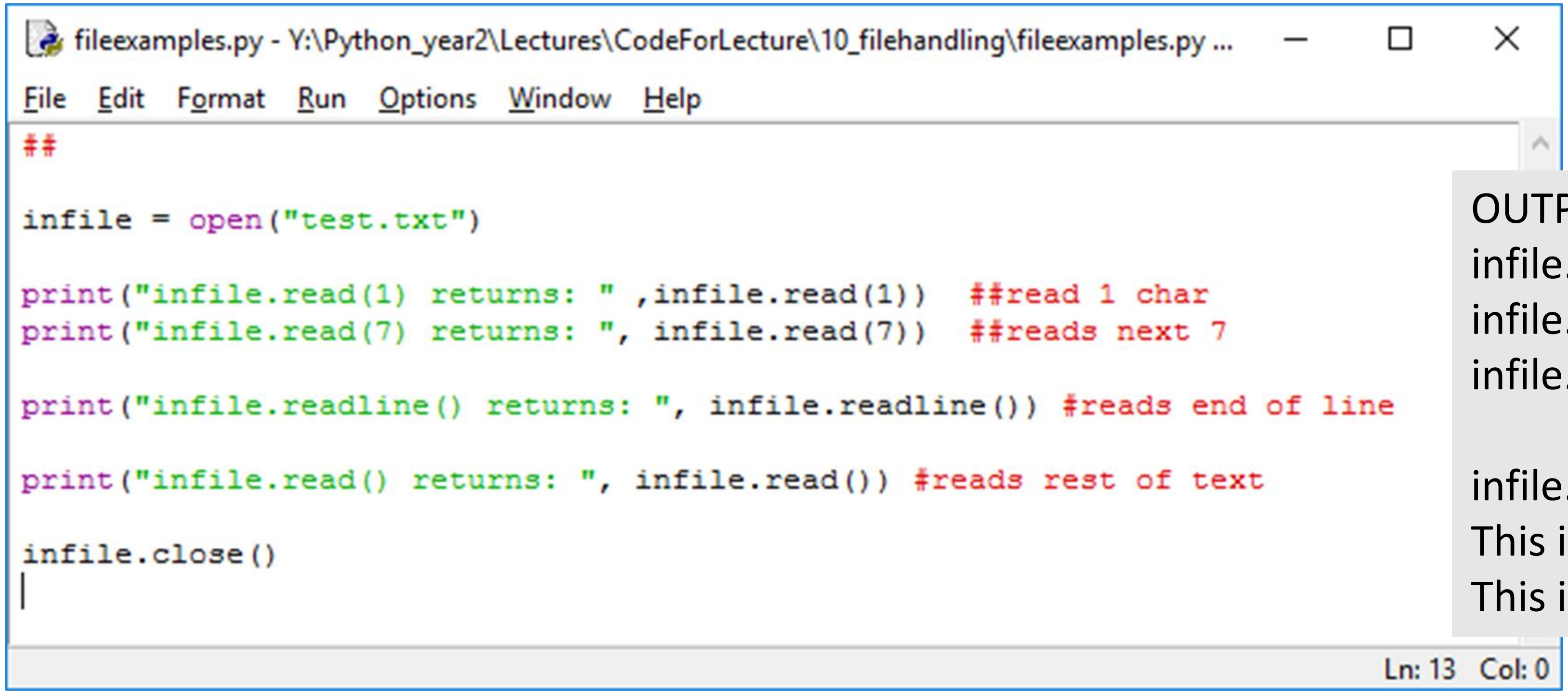
```
infile = open("mary.txt")
##readlines() reads all line up to end of file (EOF)  returns  a list
print("infile.readlines() returns all lines:\n",infile.readlines())
infile.close()
```



```
= RESTART: O:\Semester1_2021\Yr2_Python\Lectures\CodeForLectur
es\11_filehandling\ex1_fileexamples.py
infile.readlines() returns all lines:
 ['Mary had a little lamb\n', 'Her face was white as snow\n',
'and everywhere that Mary went\n', 'the lamb was sure to go']
>>> 
```



```
This is line one

This is line three, line two is blank
This is the fourth line.
```


fileexamples.py - Y:\Python_year2\Lectures\CodeForLecture\10_filehandling\fileexamples.py ...
File Edit Format Run Options Window Help
Ln: 13 Col: 0

```
##

infile = open("test.txt")

print("infile.read(1) returns: " ,infile.read(1))  ##read 1 char
print("infile.read(7) returns: ", infile.read(7))  ##reads next 7

print("infile.readline() returns: ", infile.readline()) #reads end of line

print("infile.read() returns: ", infile.read()) #reads rest of text

infile.close()
```

```
OUTPUT
infile.read(1) returns:  T
infile.read(7) returns:  his is
infile.readline() returns:  line one

infile.read() returns:
This is line three, line two is blank
This is the fourth line.
```

- `read()` Reads everything up to EOF

  ```
  infile.read()
  ```

- `read(n)` reads a specific no of characters - returns *n* characters as a string

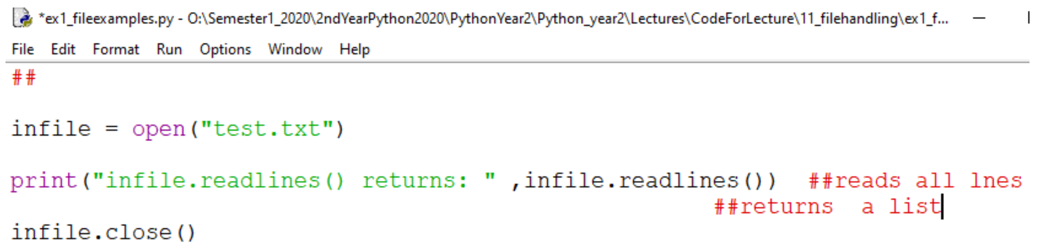
  ```
  infile.read(1) ##read next 1 chars
  infile.read(7)  ##read next 7 chars
  ```

- `readline()` read chars up to the end of the line the `\n` char, or the end of the file and cursor points to start of next line

  ```
  infile.readline()
  ```

- Use `readlines()` to return all the lines in a file as a list of lines

  ```
  infile.readlines()
  ```


*ex1_fileexamples.py - O:\Semester1_2020\2ndYearPython2020\PythonYear2\Python_year2\Lectures\CodeForLecture\11_filehandling\ex1_f...
File Edit Format Run Options Window Help

```
##

infile = open("test.txt")

print("infile.readlines() returns: " ,infile.readlines())  ##reads all lnes
                                                           ##returns  a list
infile.close()
```

```
['This is line one\n', '\n', 'This is line three, line two is
blank\n', 'This is the fourth line.']
```

- Closing the file releases the file system resources

  ```
  infile.close()
  ```

Line endings

- In Python, new line char is `'\n'`
- Text file formats are platform dependant
  - MS Windows uses \r\n 2-char sequence
  - Linux/UNIX and Mac use \n

- Pythons translates platform dependant line-ends into `\n` when reading and back to platform-dependant when writing

Reading text files

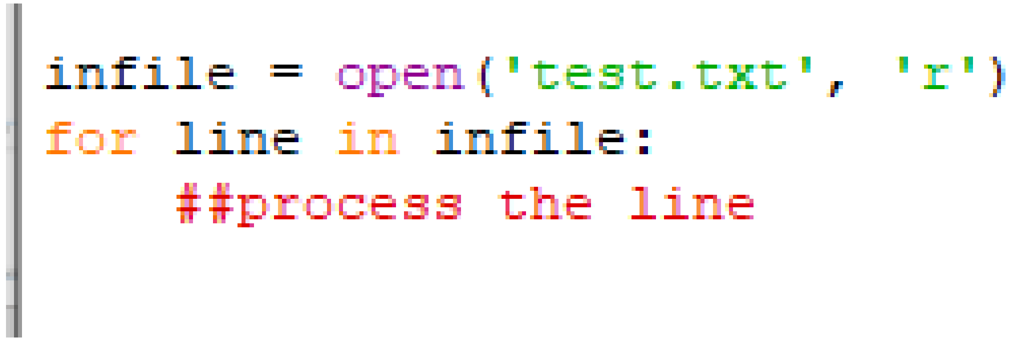
- Read the entire file into the program (and then process)

```
contentsOfFile = infile.read()
```

- Read all lines into a `list` of lines (and then process the line)

```
lineList = infile.readlines()
```

- Iterate through the file line by line process line at a time

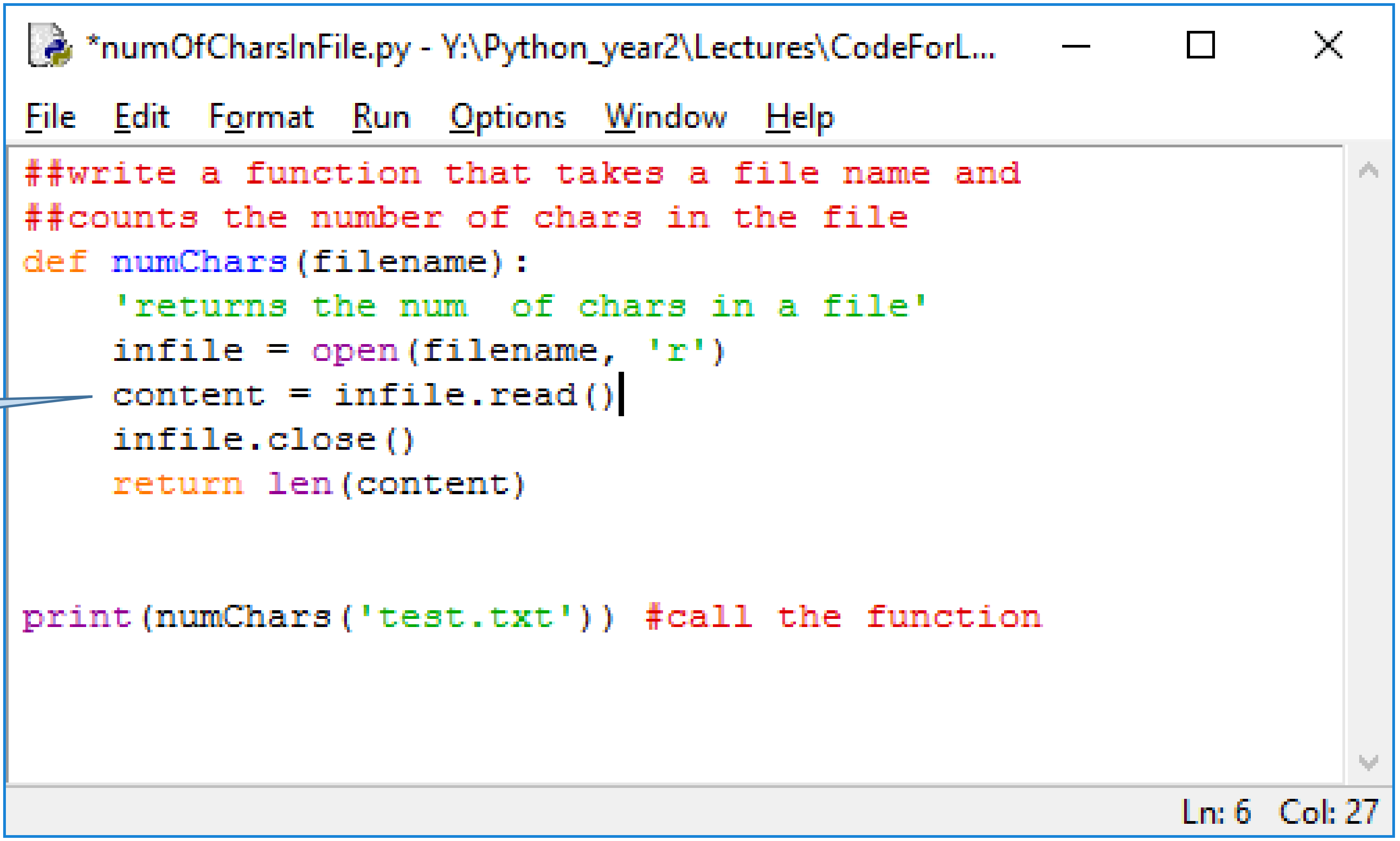


```
infile = open('test.txt', 'r')
for line in infile:
    ##process the line
```

Read contents of file and count characters

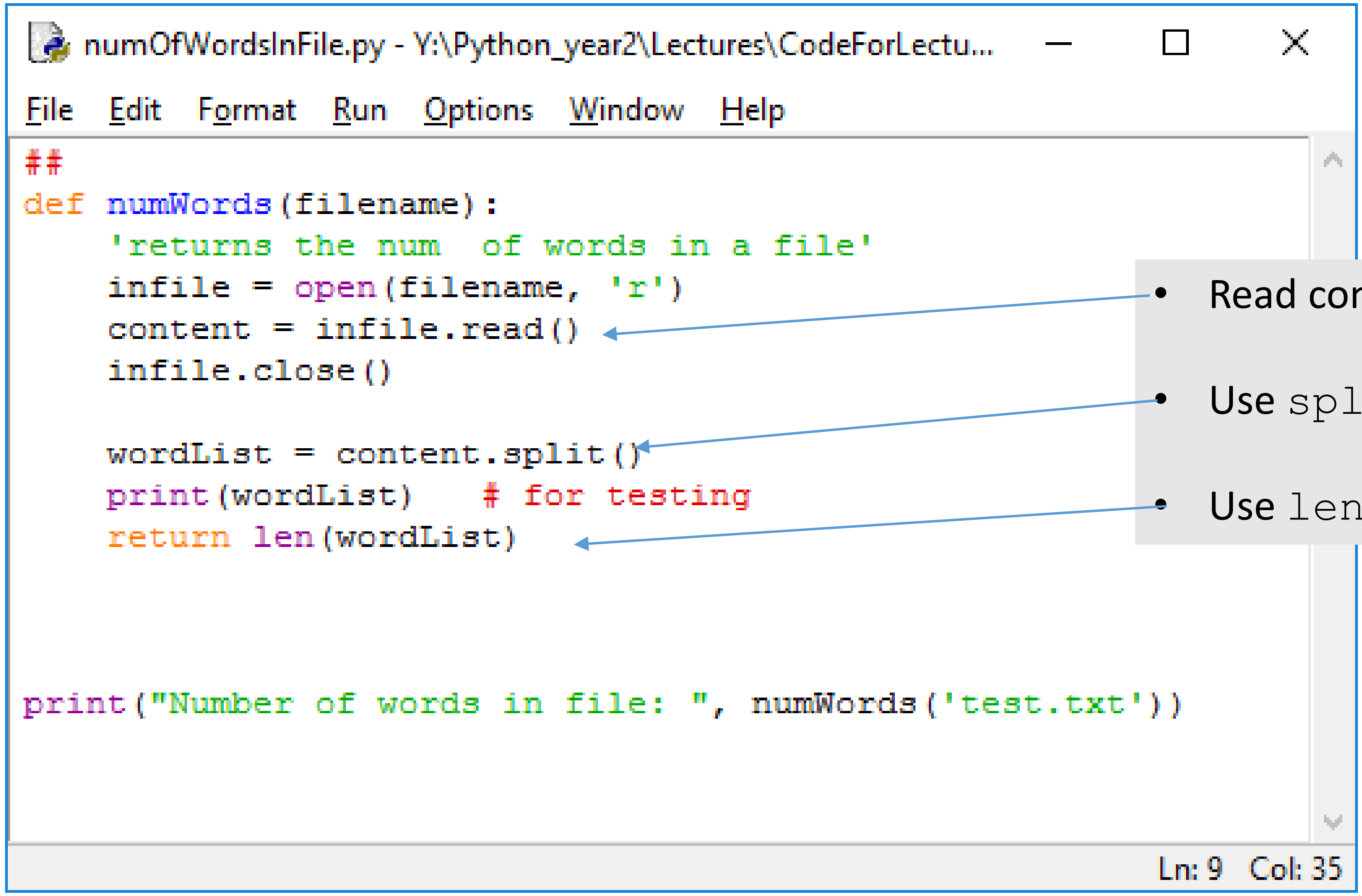
numOfCharsInFile.py

content is a string


*numOfCharsInFile.py - Y:\Python_year2\Lectures\CodeForL...
File Edit Format Run Options Window Help
Ln: 6 Col: 27

```
##write a function that takes a file name and
##counts the number of chars in the file
def numChars(filename):
    'returns the num  of chars in a file'
    infile = open(filename, 'r')
    content = infile.read()
    infile.close()
    return len(content)


print(numChars('test.txt')) #call the function
```

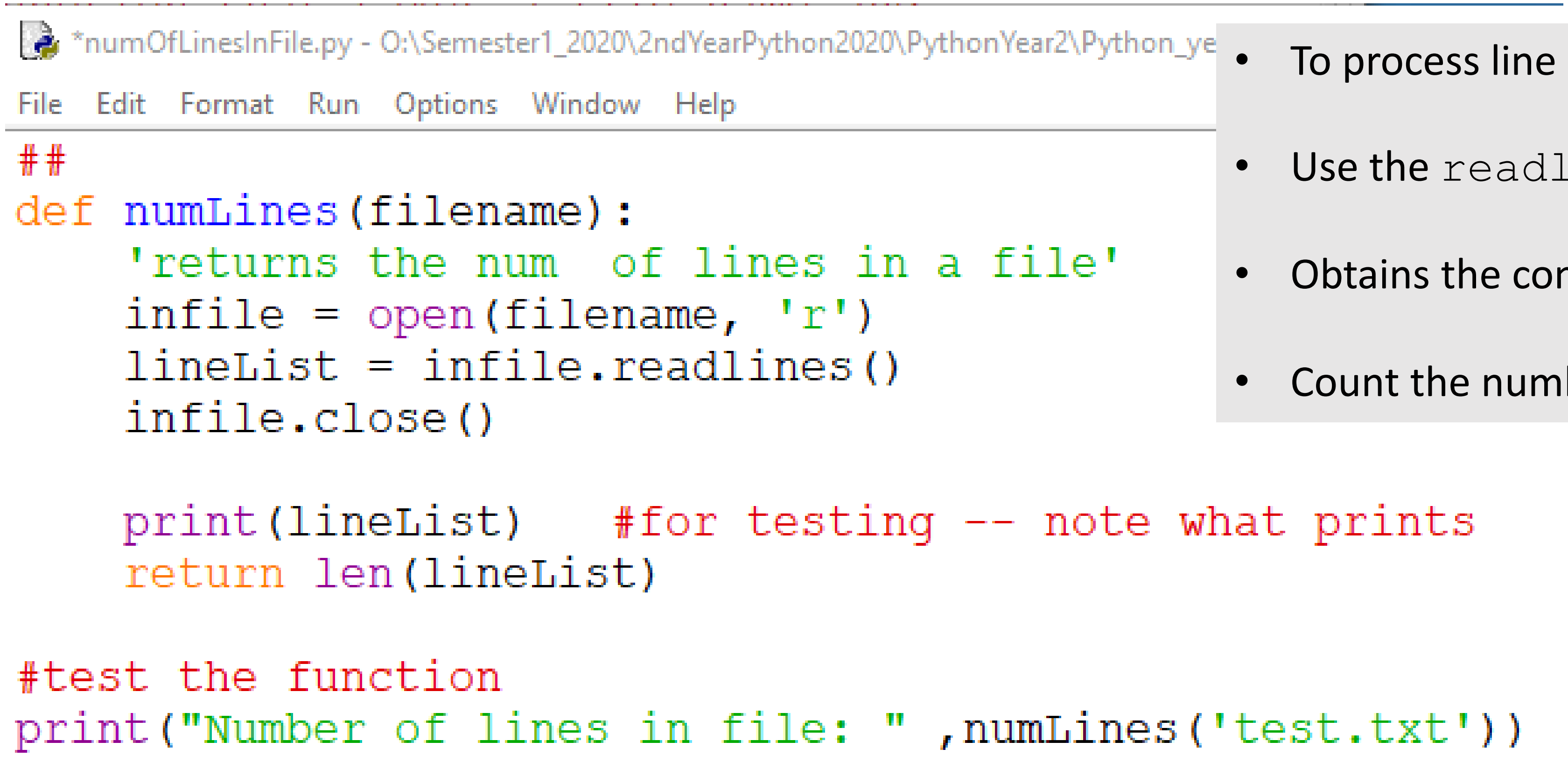
Read the contents of the file and count words


numOfWordsInFile.py - Y:\Python_year2\Lectures\CodeForLectu...
File Edit Format Run Options Window Help
Ln: 9 Col: 35

```
##
def numWords(filename):
    'returns the num  of words in a file'
    infile = open(filename, 'r')
    content = infile.read()
    infile.close()

    wordList = content.split()
    print(wordList)    # for testing
    return len(wordList)



print("Number of words in file: ", numWords('test.txt'))
```

- Read contents of file into `content`
- Use `split`() to split into list of strings
- Use `len` () to find no of words

To process line by line

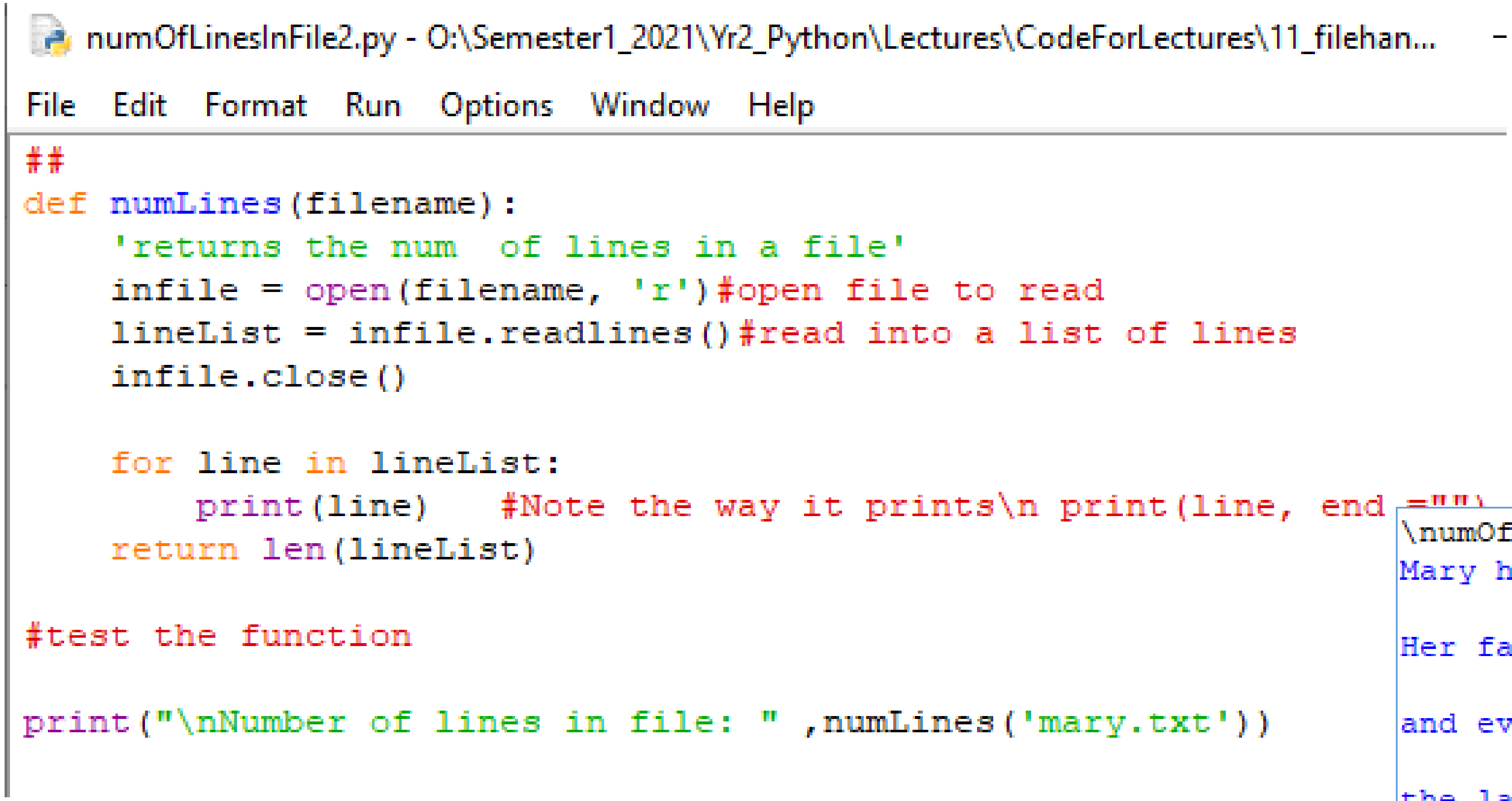


*numOfLinesInFile.py - O:\Semester1_2020\2ndYearPython2020\PythonYear2\Python_ye
File Edit Format Run Options Window Help

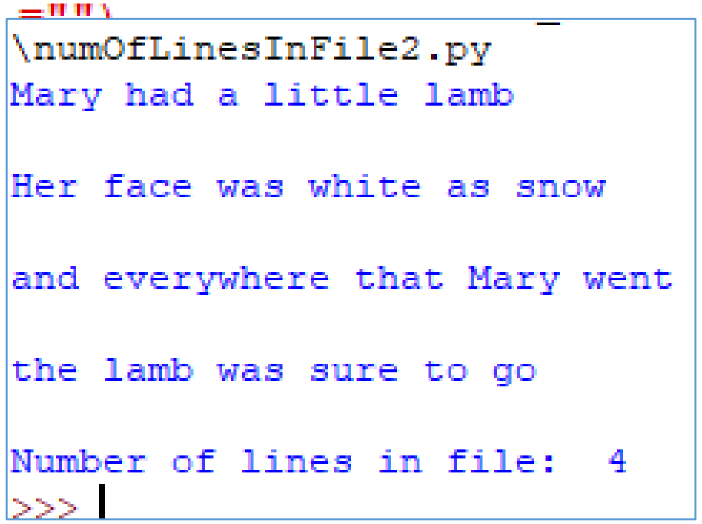
```
##
def numLines(filename):
    'returns the num  of lines in a file'
    infile = open(filename, 'r')
    lineList = infile.readlines()
    infile.close()

    print(lineList)    #for testing -- note what prints
    return len(lineList)

#test the function
print("Number of lines in file: " ,numLines('test.txt'))
```

- To process line by line
- Use the `readlines()` function
- Obtains the contents as a `list` of lines
- Count the number of lines using `len`()

To process line by line


numOfLinesInFile2.py - O:\Semester1_2021\Yr2_Python\Lectures\CodeForLectures\11_filehan...
File Edit Format Run Options Window Help

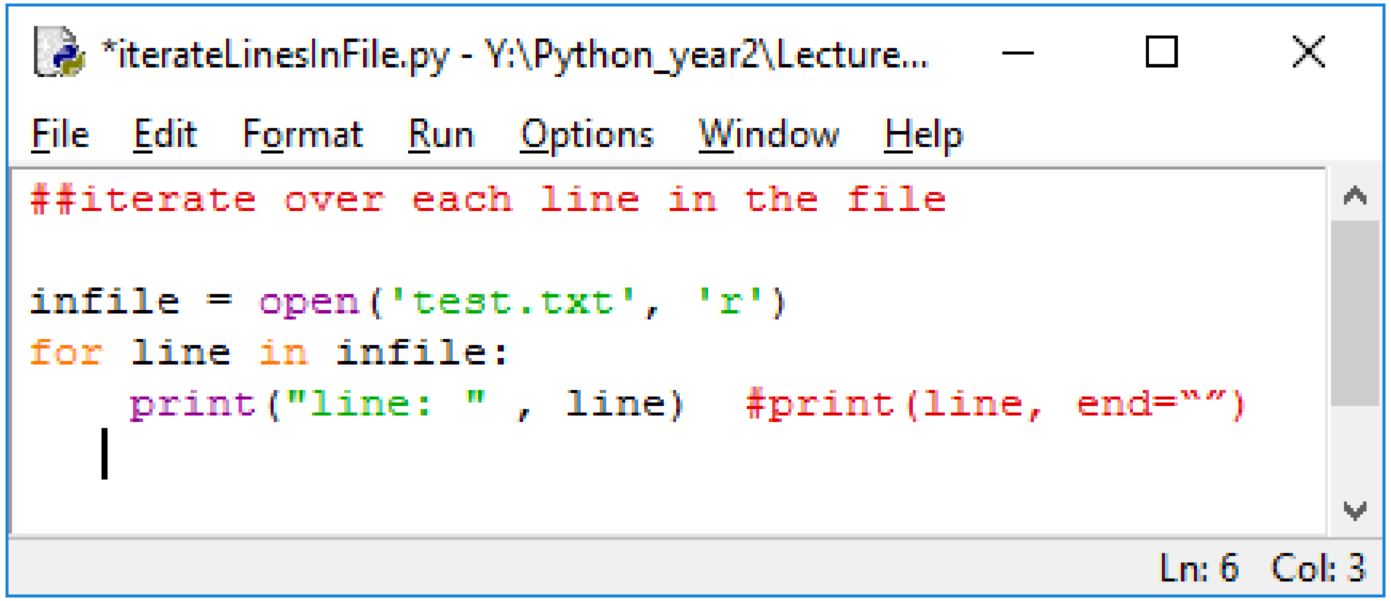
```python
##
def numLines(filename):
    'returns the num  of lines in a file'
    infile = open(filename, 'r')#open file to read
    lineList = infile.readlines()#read into a list of lines
    infile.close()

    for line in lineList:
        print(line)    #Note the way it prints\n print(line, end ="")
    return len(lineList)

#test the function

print("\nNumber of lines in file: " ,numLines('mary.txt'))
```



```
\numOfLinesInFile2.py
Mary had a little lamb

Her face was white as snow

and everywhere that Mary went

the lamb was sure to go

Number of lines in file:  4
>>> 
```

Iterate over each line in file

- If the file is very big, better to process line by line
- To iterate over each line in file


*iterateLinesInFile.py - Y:\Python_year2\Lecture...
File Edit Format Run Options Window Help
Ln: 6 Col: 3

```python
##iterate over each line in the file

infile = open('test.txt', 'r')
for line in infile:
    print("line: " , line)   #print(line, end="")
  
```

Note difference using `end=""`

- With every iteration of the for loop, `line` variable refers to next line in file
- Only one line of file in memory at any stage

Processing text in a file using `readline()`

- `readline()` reads to end of line, the `\n` char, or end of file and cursor points to start of next line.
- If the file contains a blank line, then `readline()` returns a string containing the newline character "\n".

- To repeatedly read a line of text and process it until the end of the file

```
line = infile.readline()
while line != "" :
  #Process the line.
  line = infile.readline()
```

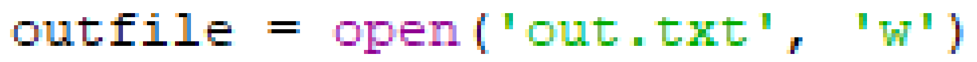
- The sentinel value is an empty string, which is returned by the `readline()` method when *end of file* has been reached.
- the `readline()` returns strings.
- If the file contains numerical data, the strings must be converted to the numerical value using the `int` or `float` function:

```
value = float(line)
```

- The newline character at the end of the line is ignored when the string is converted to a numerical value.

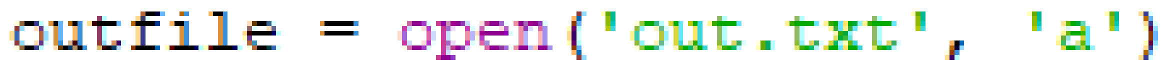
Writing to a text file

- To ***write*** to a file, open the file for writing



```
outfile = open('out.txt', 'w')
```

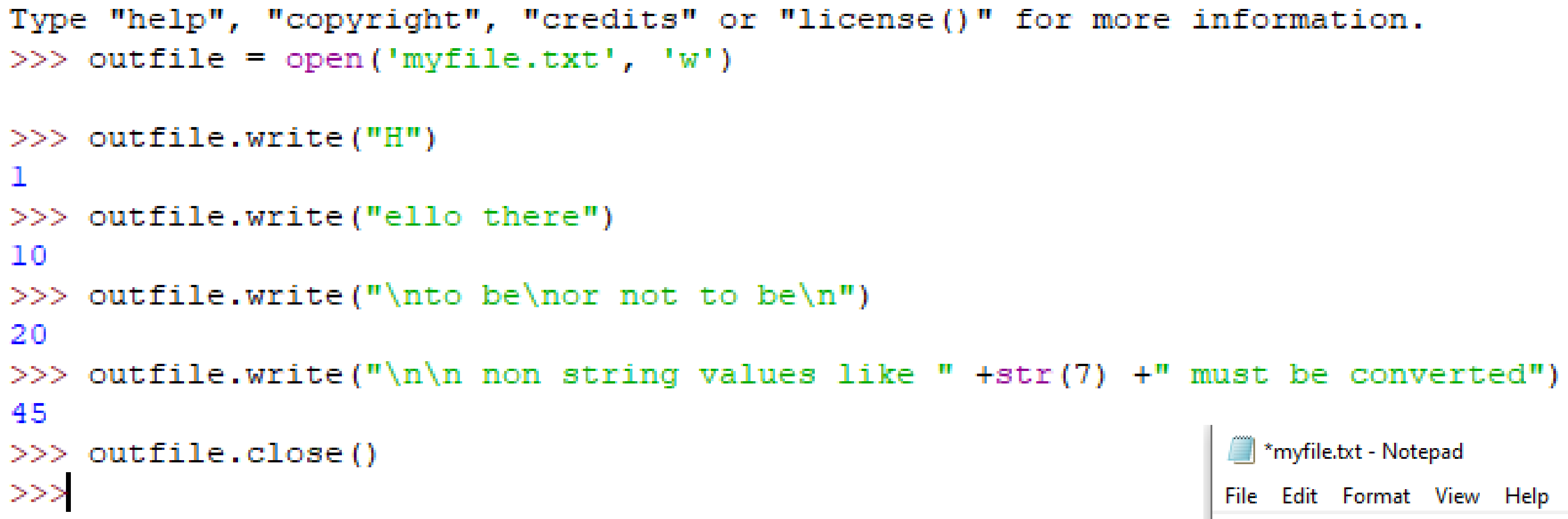
- If the file does not exist, the `open()` function will create it
- If a file exists, its contents will be erased.
- The cursor will point to beginning of empty file
- To *add* or *append* content to the end of an existing file use *mode* `‘a’`



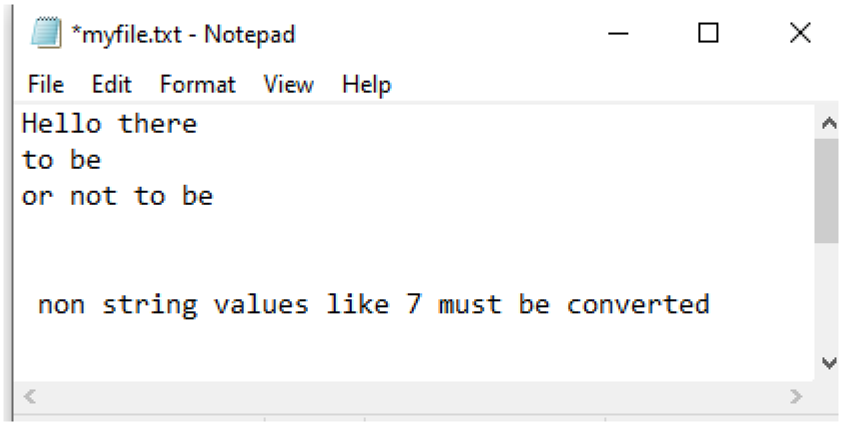
```
outfile = open('out.txt', 'a')
```

write()

- Use `write()` to write strings to the file
- `write()` returns the number of characters written to the file



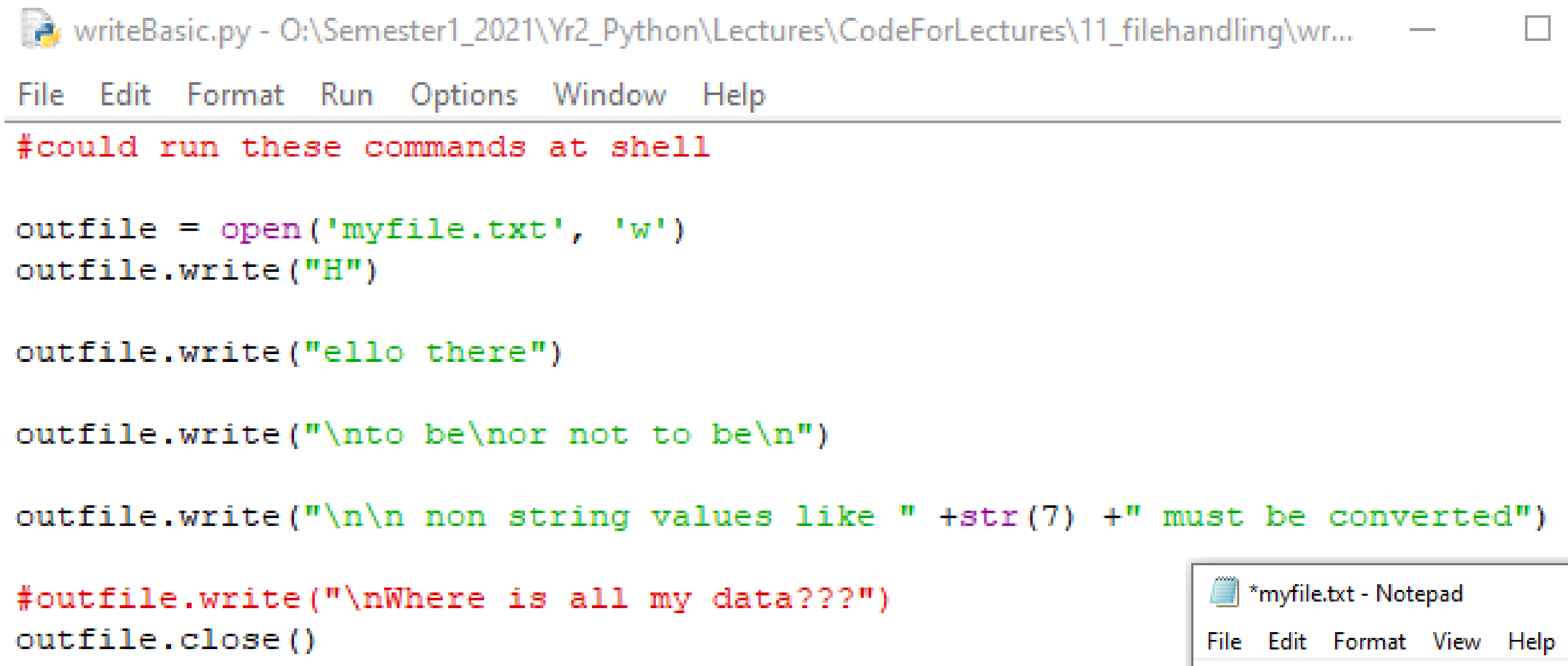
```
Type "help", "copyright", "credits" or "license()" for more information.
>>> outfile = open('myfile.txt', 'w')

>>> outfile.write("H")
1
>>> outfile.write("ello there")
10
>>> outfile.write("\nto be\nor not to be\n")
20
>>> outfile.write("\n\n non string values like " +str(7) +" must be converted")
45
>>> outfile.close()
>>>
```



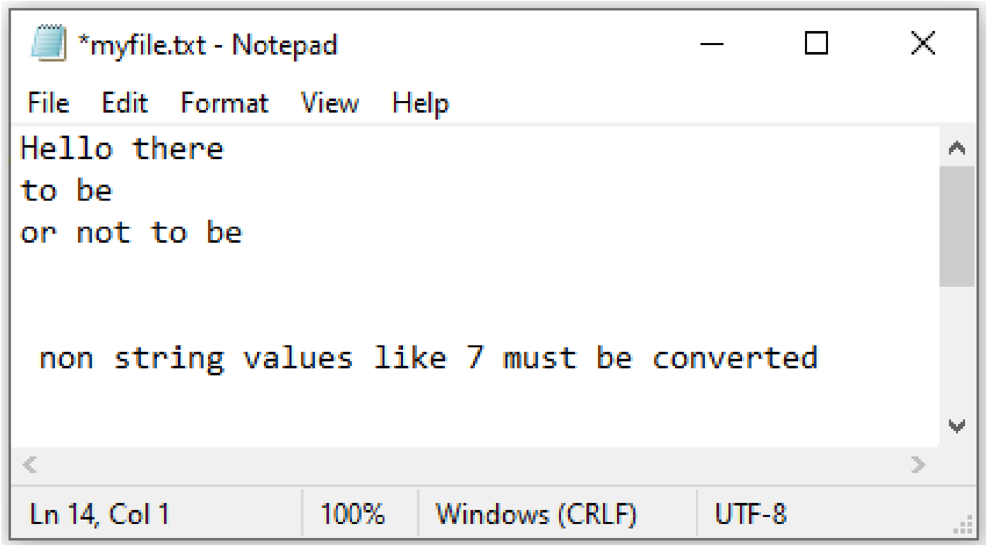
```
Hello there
to be
or not to be


 non string values like 7 must be converted
```

- MUST remember to close the file.

write()

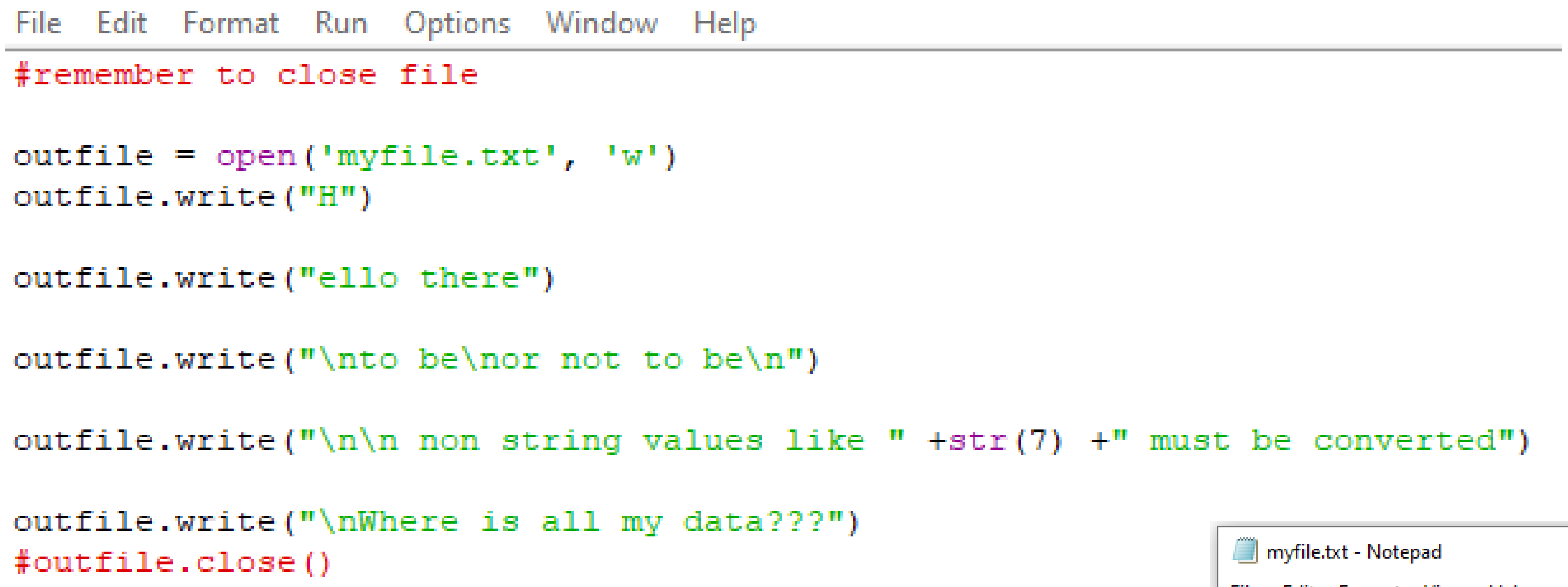

writeBasic.py - O:\Semester1_2021\Yr2_Python\Lectures\CodeForLectures\11_filehandling\wr...
File Edit Format Run Options Window Help

```
#could run these commands at shell

outfile = open('myfile.txt', 'w')
outfile.write("H")

outfile.write("ello there")

outfile.write("\nto be\nor not to be\n")

outfile.write("\n\n non string values like " +str(7) +" must be converted")

#outfile.write("\nWhere is all my data???")
outfile.close()
```

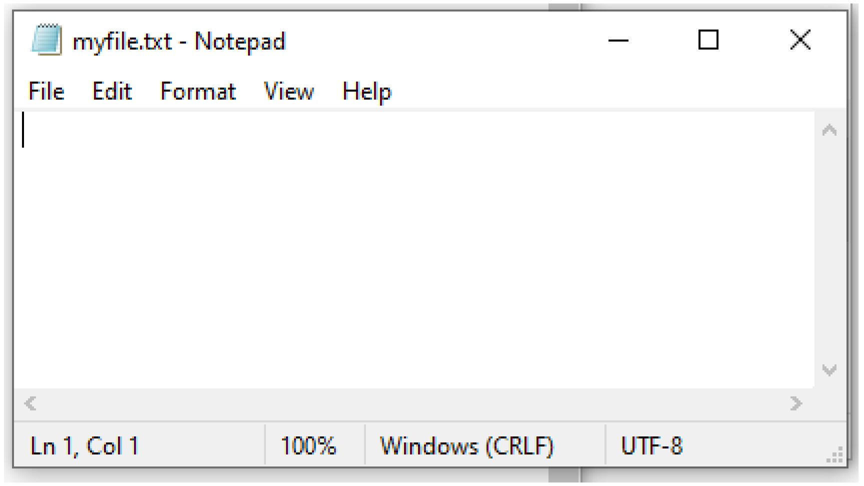


```
Hello there
to be
or not to be


 non string values like 7 must be converted
```

write()   - remember to close file



```
#remember to close file

outfile = open('myfile.txt', 'w')
outfile.write("H")

outfile.write("ello there")

outfile.write("\nto be\nor not to be\n")

outfile.write("\n\n non string values like " +str(7) +" must be converted")

outfile.write("\nWhere is all my data???")
#outfile.close()
```



Formatted output to file

- Write formatted strings to a file with the `write()` method:

```
outfile.write("No. of items: %d\nTotal: %8.2f\n" % (count, total))
```

- Use `print()` to write text to a file.
- Supply the file object as an argument with name file:

```
print("Hello, World!", file=outfile)
```

- To omit a newline, use the end argument:

```
print("Total: ", end="", file=outfile)
```

Flushing the output

- When a *file* is opened for writing a buffer is created in memory
- All writes to file are written to buffer
- Nothing is actually written to file until it is flushed
- The `close()` function will flush the files from the buffer to the file

- Or you can us `outfile.flush()`

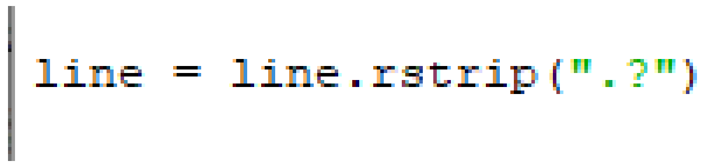
`rstrip()` method

- Use the `rstrip()` method to remove the newline character from a line of text.
- To remove the newline character, apply the `rstrip`() method to the string

```
line = line.rstrip()
```

- which results in the new string

- By default, the `rstrip()` method creates a new string in which all white space (blanks, tabs, and newlines) at the end of the string has been removed.
- To remove specific characters from the end of a string, pass a string argument containing those characters to the `rstrip`() method.
- For example, to remove a period or a question mark from the end of string



```
line = line.rstrip(".?")
```

Consider…

- Read from a file and count the occurrences of each letter in the file.
  - The uppercase letters have codes in sequential order, from 65 for the letter A through 90 for the letter Z.
  - By subtracting the code for the letter A, one obtains a value between 0 and 25 that can be used as an index to the `letterCounts` list