Lecture 10: Files

Stephen Huang

April 12, 2023

Contents

Importance

- Python allows you to read and write from/to files like most other languages.
- All the values you hold in variables will be gone at the end of your program.
- Files will outlast the program.
- Whether writing to a simple text file, reading a detailed server log, or even analyzing raw byte data (which we will NOT do), these situations require reading or writing a file.

1. Introduction

- This chapter introduces the idea of "persistent" programs that keep data in permanent storage.
- Persistent Program: they run for a long time (or all the time); they keep at least some of their data in permanent storage (a hard drive); and if they shut down and restart, they pick up where they left off.
- One of the most straightforward ways programs maintain data is by reading and writing text files.

Files

- A text file is a sequence of characters stored on a permanent medium like a hard drive.
- We consider only the text (ASCII) file in this chapter. Binary files are not considered.
- A text file is a linear structure of characters. In general, we must process input sequentially. No Random Access in this course.
- End-of-Line ("\n") characters separate a file into lines (of variable length). The implementation is OS-dependent.

Line Ending

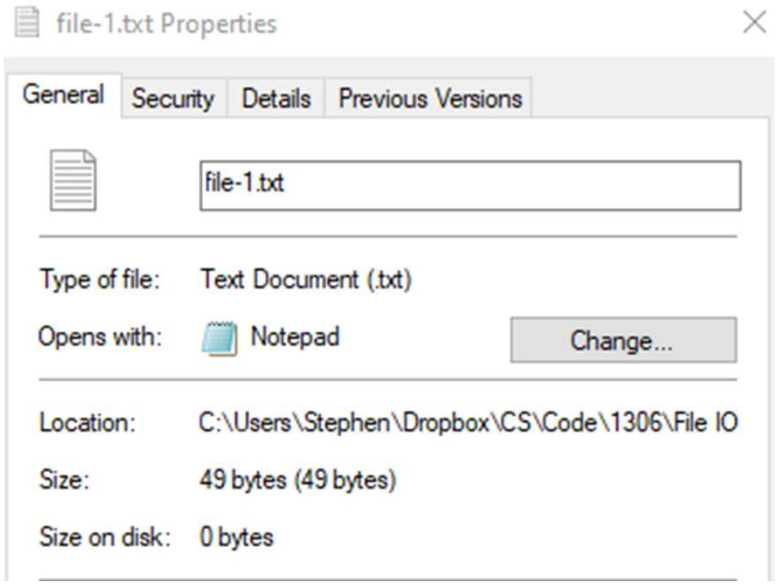
- One problem often encountered when working with file data is representing a new line or line ending.
- ASA standard states that line endings should use the sequence of the Carriage Return (CR, x0D or \r) and the Line Feed (LF, x0A, or \n) characters (CR+LF or \r\n).
- However, the ISO standard allowed for either the CR+LF characters or just the LF character.

Line Ending

- Windows uses the CR+LF (\r\n) characters, while Unix and Mac use just the LF (\n) character.
- Internally, Python uses "\n" to represent end-of-line (eoln). Technically, we use eoln's to separate lines, not to terminate lines.
- You can always add an eoln at the end of the last line to make it more uniform. But then the last line is no longer the last.
- There is no end-of-file character.

Under the Hood

```
This is a text file.
Is this clear?
That's all!
```



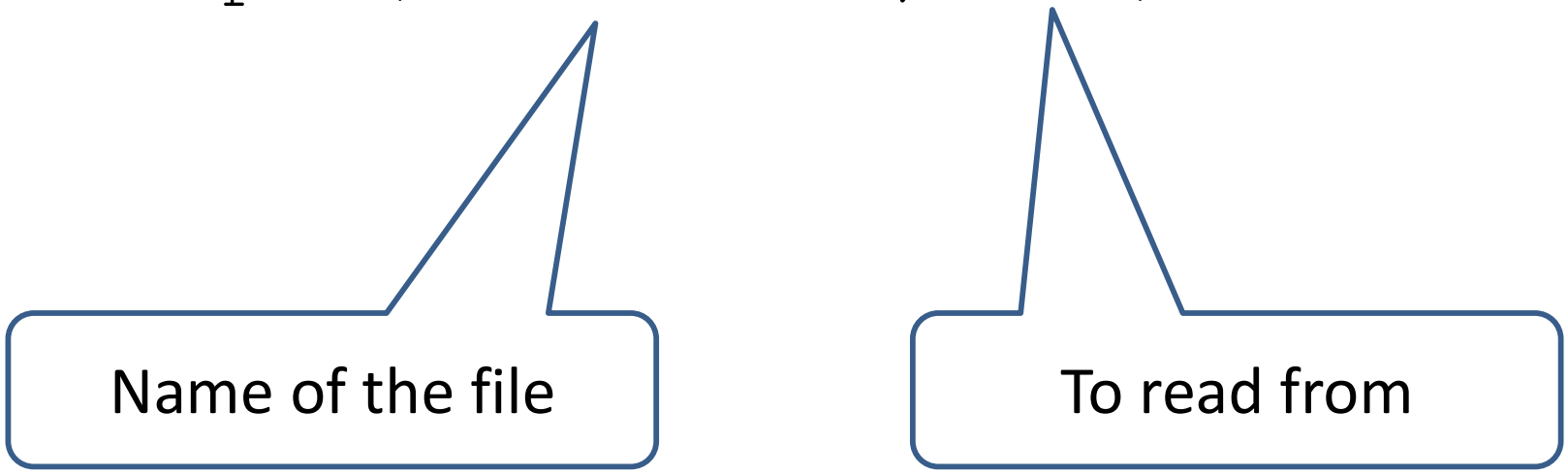

```
            0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 54 68 69 73 20 69 73 20 61 20 74 65 78 74 20 66 ; This is a text f
00000010h: 69 6C 65 2E 0D 0A 49 73 20 74 68 69 73 20 63 6C ; ile...Is this cl
00000020h: 65 61 72 3F 0D 0A 54 68 61 74 27 73 20 61 6C 6C ; ear?..That's all
00000030h: 21                                              ; !
```

ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Finding the file

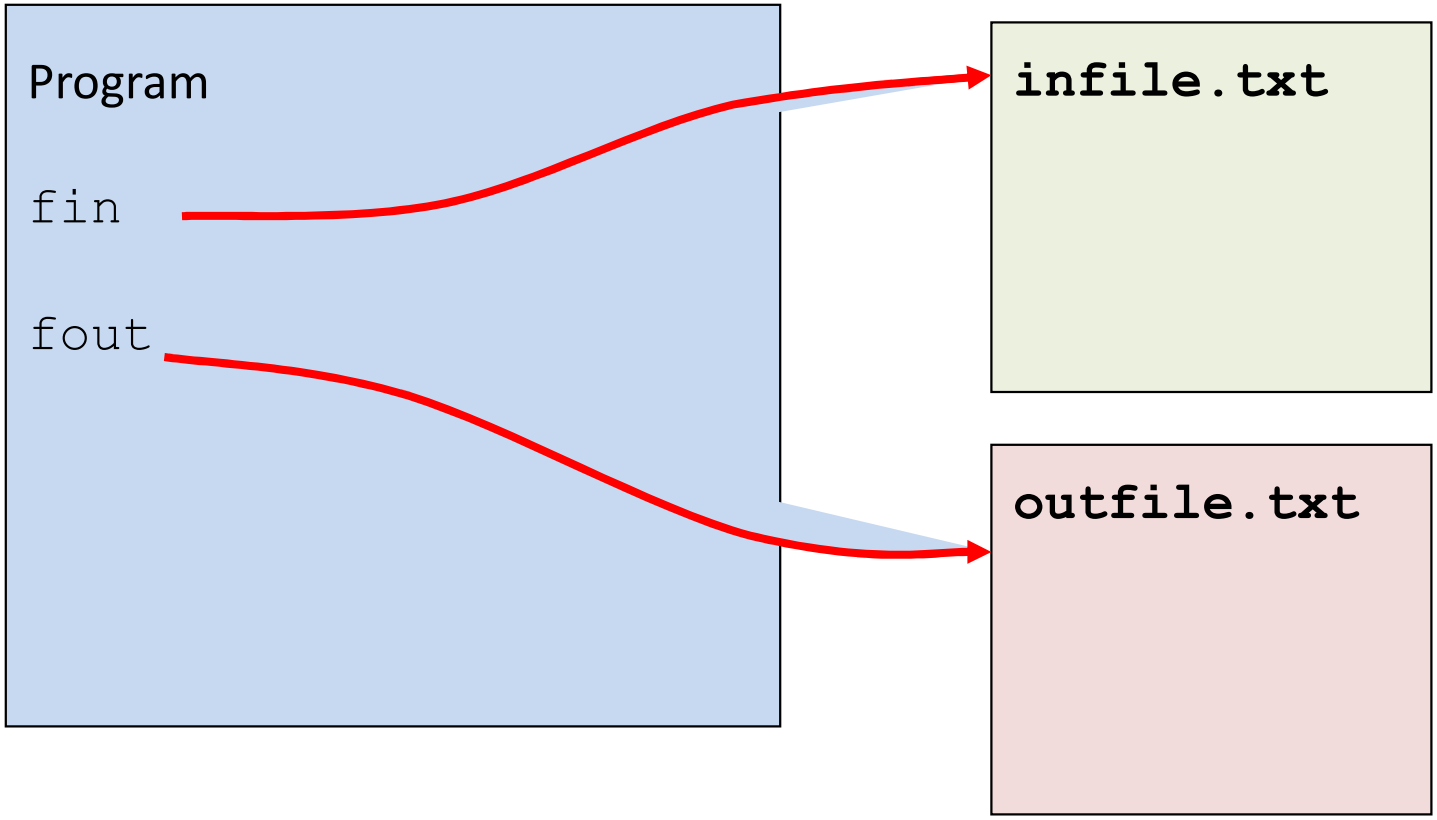
```
import os

os.chdir("C:/Users/guido/My Documents")

file = open("info.txt", "r")
```



2. Opening a file

- Stream (internal/program object) is our representation of a file (external object) in the program.
- A file must be connected to a stream before it can be used.

Opening file



File Name

- Files have two “names” in our programs
  - External file name
    - Also called "physical file name."
      - Like "infile.txt"
    - Sometimes considered "real file name."
    - Used only once in the program (to open)
      - Once connected, forget it!
  - Stream (internal) name
    - Also called "logical file name."
    - The program uses this name for all file activity

Open

- A file can be opened in two modes: read (r) or write (w).

```
fin = open('input.txt', 'r')
fout = open('output.txt', 'w')
```

- In opening a file for input, the file must exist in your directory.
  - A common error is misspelling the name of the file.
  - All subsequent input operations will fail if the file does not open.
- Trying to open a file that does not exist will fail.

Open

```
open(file_name [, access_mode][, buffering])
```

- file_name: name of the file that you want to access.
- access_mode: The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. This parameter is optional; the default file access mode is read (r).
- buffering: Not important in this chapter.

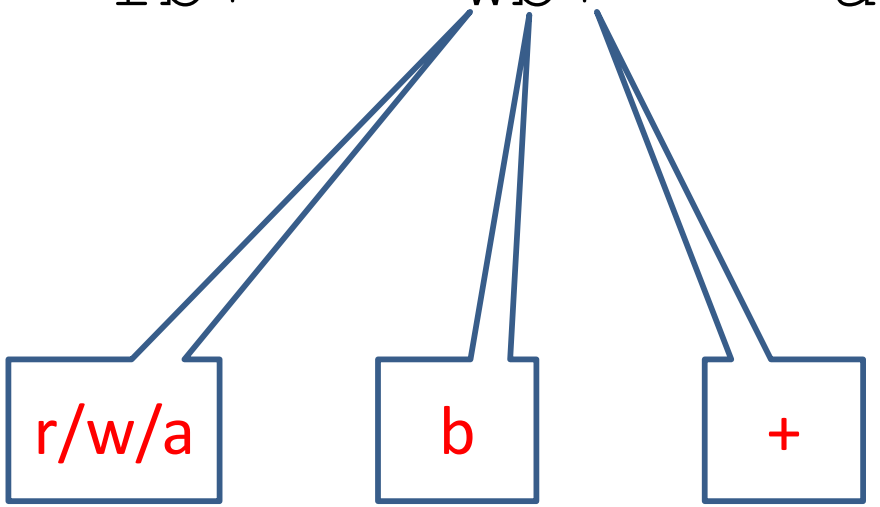
Modes

1. Operation Mode:
   - Read
   - Write
   - Append

2. Format Mode:
   - ASCII Text
   - Binary

3. R/W Option: +

- `r`
- `rb`
- `r+`
- `rb+`
- `w`
- `wb`
- `w+`
- `wb+`
- `a`
- `ab`
- `a+`
- `ab+`



Modes

- We won't discuss binary files here.
- You should know the three basic modes of `r`, `w`, and `a` first.
- Then try to understand the three corresponding `+` modes later.

Modes

Mode	read	write	append	pointer	create	truncate
r	√			B		
w		√		B	√	
a		*	√	E	√	
r+	√	√		B		
w+	√	√		B	√	√
a+	*	*	√	E	√	

* If file does not exist, create a new file for r/w

B = beginning,

E = End

Close()

- An opened file will be closed automatically at the end of the program.
- However, it is a good programming practice to close a file if it is no longer needed.
- If so, you can reuse the file name, i.e., open another file using the name.

With Open As

- Using the following syntax will limit the scope of the file to be inside the "with" clause.

```
with open("test.txt") as f:
    # read from the file
    # f closed at the end of this block
```

- Compare to:

```
f = open("test.txt")
# read from the file
f.close()
```

With .. As

- The with statement initiates a context manager.
- The context manager opens a file and manages the file resource as long as the context is active.
- All the code in the indented block depends on the file object being open.
- Once the indented block ends or raises an exception, the file will close automatically.
- Otherwise, you must close the file explicitly.

Closing Files is Important

- An opened file requires the use of certain system resources.
- The leaking of resources may be due to poor programming practice or a malicious program attacking the system.
- Operating systems limit the number of open files a single process can have.
- Keeping files open leaves you vulnerable to losing data when the system crashes.

3. Reading from a file

- There are multiple ways to read from a file.
  a. Using an Iterator
  b. Readline()
  c. Read()
  d. Readlines()

One line at a time

All lines in one call

- Some do it one line at a time, and others read the whole file.
- Storage considerations for large files. Do you need the entire file altogether?

Input from a file

- A text file is divided into lines of various lengths. An end-of-line character (\n) is used as a separator.
- An extra end-of-line may occur when printing the string because of the \n character at the end of each line.
- End-of-line can be removed by the strip() method of string.

(a) Using an Iterator

```
for line in f:
```

- Reads one line from the file per iteration.
- Returns the line.
- The line includes the end-of-line character (if any).
- The iterator goes from the first line to the last line of the file.
- There is no risk of reading past the end of the file.

Example

```
fin = open('test.txt')
for line in fin:
    print(line)
fin.close()
```

Remove the eoln

```
fin = open("test.txt")
for line in fin:
    print(len(line))
    line = line.strip()
    print(line)
    print(len(line), "\n")
```

```
24
This is the first line.
23
```

(b) Readline

```
line = f.readline()
```

- Reads one line from the file.
- Returns the line.
- The line includes the eoln character.
- Use a (while) loop to read all lines.
- Reading past the eoln returns an empty string. An empty string is considered False as a Boolean value that can be used to terminate a while loop.

Example

```
f = open('test.txt')
line = f.readline()
while line:
    print(line)
    line=f.readline()
f.close()
```

Example

```
f = open('test.txt')
line = 'none empty'
while line:
    line = f.readline()
    print(line)
f.close()
```

This program prints one extra empty line if no `eoln` at the end.

How do we fix that?

A comparison

Iterator

```
f = open('test.txt')

for line in f:
    print(line)


f.close()
```

readline()

```
f = open('test.txt')
line = f.readline()
while line:
    print(line)
    line = f.readline()

f.close()
```

(c) Read

```
line = f.read()
```

- Reads the whole file as one string.
- Returns the string.
- The string includes all eoln characters.
- There may be no eoln character at the end of the last line.

Example

```
f = open('test.txt')
lines = f.read()
print(lines)
f.close()
```

(d) Readlines

```
lines = f.readlines()
```

- Reads the whole file into a string and then splits it into a list of separate lines.
- Return a list of lines.
- Each line in the list contains an end-of-line character.*

Example

```
f = open('test.txt')
lines = f.readlines()
print(lines)
f.close()
```

A comparison

read()

```
f = open('test.txt')
lines = f.read().splitlines()
for line in lines:
    print(line)
f.close()
```

readlines()

```
f = open('test.txt')
lines = f.readlines()
for line in lines:
    print(line)
f.close()
```

Another comparison

readline()

```
f = open('test.txt')
line = " "
while line:
    line = f.readline()
    print(line)
f.close()
```

readlines()

```
f = open('test.txt')
lines = f.readlines()
for line in lines:

    print(line)
f.close()
```

A Grand Comparison

		returns		
method	reads	type	end-of-line	Loop
Iterator	one line	string	with	for
readline()	one line	string	with	while
read()	the file	string	with all	-
readlines()	the file	list of strings	with all	-

Closing the file

- fin.close() will close the file fin.
- A file that is closed will not be accessible.
- The file object, fin for this case, can be reused.

```
fin = open("test1.txt")
for line in fin:
    line = line.strip()
    print(line)
print("---")
fin.close()
fin = open("test2.txt")
for line in fin:
    line = line.strip()
    print(line)
print("---")
```

Close

- If you must read a file multiple times, you can close it using the close() method and then open it again using open().
- There is no need to close() the file before issuing the second open() because if a file is already open, Python will close it before opening it again.
- However, it is a good practice to close files when done with them.

Exception Handling

```
import sys
try:
    f = open('integers.txt')
    s = f.readline()
    i = int(s.strip())
except IOError as e:
    errno, strerror = e.args
    print(f"I/O error({errno}): {strerror}")
except ValueError:
    print("No valid integer in line.")
except:
    print("Unexpected error:", sys.exc_info()[0])
    raise
```

Exception Template

```
try:
    logFile = 'log.txt'
    report = open(logFile, 'w')
    report.write('some message')

except Exception as e:
    report.write('an error message')

finally:
    report.close()
```

Failed Attempt

```
fin = open("test.txt")
for line in fin:
    line = line.strip()
    print(line)
print("---")
fin.close()
for line in fin:
    line = line.strip()
    print(line)
```

Typical Steps

```
fin = open("test.txt")  # open
for line in fin:        # get line
    line = line.strip()# strip
    print(line)         # process
fin.close()             # close
```

With Open As

```
with open("test.txt") as f:
    for line in f:
        print(line)
# File f is closed when existing the with
```

- File f is closed automatically, but variable f survived.
- It is good practice to use the with keyword when dealing with file objects.
- There is a Boolean variable `f.closed`.

4. Writing to a file

- Use open with '`w`' mode to write to a file.
- If the file already exists, opening it in write mode clears the old data and starts fresh, so be careful!
- If the file doesn't exist, a new one is created for writing.

Write

- After opening the file, we can use the `write()` or `writelines()` method to write to it.
- Additionally, an argument exists for the `print()` function that names the output file.
- The `write()` method takes a single argument: the string to be written to the file.
- The `writelines()` method writes the list items to the file.
- The `write()` method is less flexible than the `print()` function.

Write()

```
fin  = open("test.in",  'r')
fout = open("test.out", 'w')
count = 0
for line in fin:
    count += 1
    fout.write(f"{count:03d}: {line}")
fin.close()
fout.close()
```

```
001: hello1
002: ok2
003: byebye3
```

Write()

```
f = open("mytest.txt", 'w')
a = 1.0
b = "Hello"
c = [1, 2, 3]

f.write(f'{a}, {b}, {c}')
f.close()
```

1.0, Hello, [1, 2, 3]

Print()

```
f = open("mytest.txt", 'w')
a = 1.0
b = "Hello"
c = [1, 2, 3]

print(a, b, c, file=f)
f.close()
```

1.0, Hello, [1, 2, 3]

Writelines()

```
f1 = open("out1.txt",'w')
f2 = open("out2.txt",'w')
values1 =["one","two","three"]
values2 =["one\n","two\n","three\n"]
f1.writelines(values1)
f2.writelines(values2)
```

5. Appending to a file

- If we write to a file in append mode, the new text is appended to the end of the file.

```
f = open("test.in", 'a')
f.write("\nThis is the last line")
f.close()
```

Try Write

```
try:
    logFile = 'log.txt'
    log = open(logFile, 'a')
    print('some message', file=log)
except Exception as e:
    print('an error message', file=log)
finally:
    log.close()
```

6. Modules

- Any Python code file can be imported as a module to other Python code.
- Module allows multiple people and programs to work on one project, sharing the same code.
- The module name is the file name without the extension. To import a module x.py, say "import x"

Import

mycount.py

```
def linecount(filename):
    count = 0
    for line in open(filename):
        count += 1
    return count
```

```
import mycount as mc:
print(mc.linecount('test.txt'))
```