

CS 104

Introduction to Programming

File I/O

Lecture Overview

- File I/O
 - Basics of Files
 - File Access Modes
 - Sequential Access
 - Random Access

Disclaimer: Much of the material and slides for this lecture were borrowed from

- Ruth Anderson, Michael Ernst and Bill Howe's University of Washington CSE 140 class,
- Ana Bell, Eric Grimson, John Guttag's MIT 6.0001 class
- Keith Levin's University of Michigan STATS 507 class

Lecture Overview

- File I/O
 - Basics of Files
 - File Access Modes
 - Sequential Access
 - Random Access

Persistent Data

- So far, we only know how to write “transient” programs
 - Data disappears once the program stops running
- Files allow for persistence
 - Work done by a program can be saved to disk... ..and picked up again later for other uses.
- Examples of persistent programs:
 - Operating systems
 - Databases
 - Servers

Key idea: Program information is stored permanently (e.g., on a hard drive), so that we can start and stop programs without losing **state** of the program (values of variables, where we are in execution, etc).

Reading and Writing Files

Underlyingly, every file on your computer is just a string of bit...

0	1	1	0	0	0	1	1	0	1	1	0	0	0	0	1	0	1	1	1	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

...which are broken up into (for example) bytes...

0	1	1	0	0	0	1	1	0	1	1	0	0	0	0	1	0	1	1	1	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

...groups of which correspond (in the case of text) to characters.

0	1	1	0	0	0	1	1	0	1	1	0	0	0	0	1	0	1	1	1	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

c

a

t

Files and Filenames

- A **file** object represents data on your disk drive
 - Can read from it and write to it
- A **filename** (usually a string) states where to find the data on your disk drive
 - Can be used to find/create a file
- Each operating system comes with its own file system for creating and accessing files:
 - Linux/Mac: `"/home/rea/cs104/lectures/file_io.pptx"`
 - Windows: `"C:\Users\rea\MyDocuments\cute_dog.jpg"`

Two Types of Filenames

- An Absolute filename gives a specific location on disk:
"/home/rea/cs104/14wi/lectures/file_io.pptx" or
"C:\\Users\\rea\\MyDocuments\\homework3\\images\\Husky.png"
 - Starts with "/" (Unix) or "C:\" (Windows)
 - Warning: code will fail to find the file if you move/rename files or run your program on a different computer
- A Relative filename gives a location relative to the current working directory:
"lectures/file_io.pptx" or "images\\Husky.png"
 - Warning: code will fail to find the file unless you run your program from a directory that contains the given contents
- *A relative filename is usually a better choice*

Examples

Linux/Mac: These could all refer to the same file:

`"/home/rea/class/cs104/homework3/images/Husky.png"`

`"homework3/images/Husky.png"`

`"images/Husky.png"`

`"Husky.png"`

Windows: These could all refer to the same file:

`"C:\Users\rea\My Documents\class\cs104\homework3\images\Husky.png"`

`"homework3\images\Husky.png"`

`"images\Husky.png"`

`"Husky.png"`

Depending on what your current working directory is
`$ pwd -> print working directory`

Locating files: the os module

```
>>> import os
>>> cwd = os.getcwd()
>>> cwd
'/Users/r2d2/'

>>> os.listdir()
['death_star_plans', 'princess_leia']

>>> os.listdir('princess_leia')
['Obi-Wan.txt', 'Anakin.txt']

>>> os.chdir('princess_leia')
>>> cwd
'/Users/r2d2/princess_leia'
```

Locating files: the os module

```
>>> import os
>>> cwd = os.getcwd()
>>> cwd
'/Users/r2d2/'
```

os module lets us interact with the operating system.
<https://docs.python.org/3.6/library/os.html>

os.getcwd() returns a string corresponding to the **current working directory**.

```
>>> os.listdir()
['death_star_plans', 'princess_leia']
```

```
>>> os.listdir('princess_leia')
['Obi-Wan.txt', 'Anakin.txt']
```

os.listdir() lists the contents of its argument, or the current directory if no argument.

```
>>> os.chdir('princess_leia')
>>> cwd
'/Users/r2d2/princess_leia'
```

os.chdir() changes the working directory. After calling `chdir()`, we're in a different cwd.

Locating files: the os module

```
>>> import os
>>> cwd = os.getcwd()
>>> cwd
'/Users/r2d2/'

>>> os.listdir()
['death_star_plans', 'princess_leia']

>>> os.listdir('princess_leia')
['c3po', 'Obi-Wan.txt', 'Anakin.txt']

>>> os.path.abspath('princess_leia/Obi-Wan.txt')
'/Users/r2d2/princess_leia/Obi-Wan.txt'
```

Locating files: the os module

```
>>> import os
>>> cwd = os.getcwd()
>>> cwd
'/Users/r2d2/'
```

```
>>> os.listdir()
['death_star_plans', 'princess_leia']
```

```
>>> os.listdir('princess_leia')
['c3po', 'Obi-Wan.txt', 'Anakin.txt']
```

Use os.path.abspath to get the absolute path to a file or directory.

```
>>> os.path.abspath('princess_leia/Obi-Wan.txt')
'/Users/r2d2/princess_leia/Obi-Wan.txt'
```

Locating files: the os module

```
>>> import os
>>> os.chdir('/Users/r2d2')
>>> os.listdir('princess_leia')
['c3po', 'Obi-Wan.txt', 'Anakin.txt']

>>> os.path.exists('princess_leia/Anakin.txt')
True

>>> os.path.exists('princess_leia/JarJarBinks.txt')
False

>>> os.path.isdir('princess_leia/c3po')
True

>>> os.path.isdir('princess_leia/Obi-Wan.txt')
False
```

Locating files: the os module

```
>>> import os
>>> os.chdir('/Users/r2d2')
>>> os.listdir('princess_leia')
['c3po', 'Obi-Wan.txt', 'Anakin.txt']
```

```
>>> os.path.exists('princess_leia/Anakin.txt')
True
```

Check whether or not a file/directory exists.

```
>>> os.path.exists('princess_leia/JarJarBinks.txt')
False
```

```
>>> os.path.isdir('princess_leia/c3po')
True
```

Check whether or not this is a directory.
os.path.isfile() works analogously

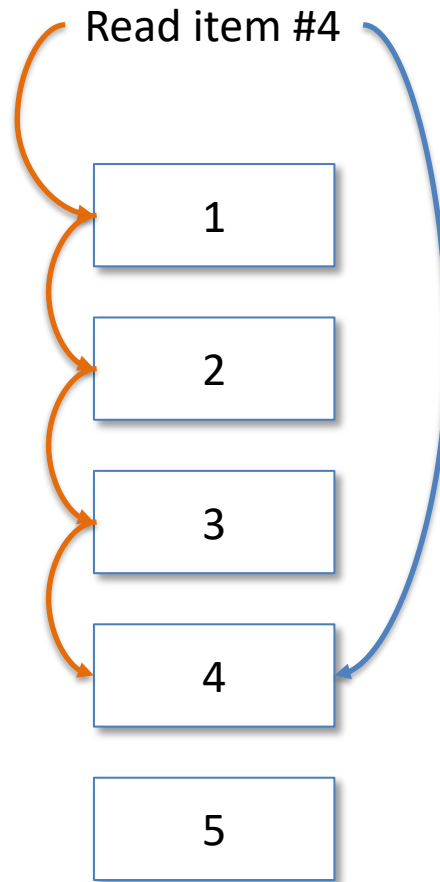
```
>>> os.path.isdir('princess_leia/Obi-Wan.txt')
False
```

Lecture Overview

- File I/O
 - Basics of Files
 - File Access Modes
 - Sequential Access
 - Random Access

Sequential vs. Random Access

Reading/Writing
data consequently.



Reading/Writing
data directly.

Lecture Overview

- File I/O
 - Basics of Files
 - File Access Modes
 - Sequential Access
 - Random Access

Reading files

```
emre:~/demo$ cat demo.txt  
This is a demo file.  
It is a text file, containing three lines of text.  
Here is the third line.  
emre:~/demo$
```

```
>>> f = open('demo.txt')  
>>> type(f)  
<type 'file'>
```

```
>>> f.readline()  
'This is a demo file.\n'
```

Reading files

This is the command line. We'll see lots more about this later, but for now, it suffices to know that the command `cat` prints the contents of a file to the screen.

```
emre:~/demo$ cat demo.txt
This is a demo file.
It is a text file, containing three lines of text.
Here is the third line.
emre:~/demo$
```

```
>>> f = open('demo.txt')
>>> type(f)
<type 'file'>
```

Open the file `demo.txt`. This creates a file object `f`
<https://docs.python.org/3/glossary.html#term-file-object>

```
>>> f.readline()
'This is a demo file.\n'
```

Provides a method for reading a single line from the file. The string `'\n'` is a **special character** that represents a new line. More on this soon.

Reading files

```
emre:~/demo$ cat demo.txt
This is a demo file.
It is a text file, containing three lines of text.
Here is the third line.
emre:~/demo$
```

```
>>> f = open('demo.txt')
>>> f.readline()
'This is a demo file.\n'

>>> f.readline()
'It is a text file, containing three lines of text.\n'

>>> f.readline()
'Here is the third line.\n'

>>> f.readline()
```

Reading files

```
emre:~/demo$ cat demo.txt
This is a demo file.
It is a text file, containing three lines of text.
Here is the third line.
emre:~/demo$
```

```
>>> f = open('demo.txt')
```

```
>>> f.readline()
'This is a demo file.\n'
```

Each time we call `f.readline()`,
we get the next line of the file...

```
>>> f.readline()
'It is a text file, containing three lines of text.\n'
```

```
>>> f.readline()
'Here is the third line.\n'
```

```
>>> f.readline()
```

...until there are no more lines to read, at
which point the `readline()` method returns
the empty string whenever it is called

Reading files

```
>>> f = open('demo.txt')
>>> for line in f:
...     for wd in line.split():
...         print(wd.strip('.,'))
```

We can treat f as an iterator, in which each iteration gives us a line of the file.

Iterate over each word in the line (splitting on ' ' by default).

Remove the trailing punctuation from the words of the file.

```
This
is
a
demo
file
It
is
a
text
file
containing
three
lines
of
text
Here
is
the
third
line
```

open () provides a bunch more (optional) arguments,
some of which we'll discuss later.
<https://docs.python.org/3/library/functions.html#open>

Reading files

```
>>> with open('demo.txt') as f:  
...     for line in f:  
...         for wd in line.split():  
...             print(wd.strip('.,'))
```

This
is
a
demo
file
It
is
a
text
file
containing
three
lines
of
text
Here
is
the
third
line

You may often see code written this way, using the with keyword. It suffices to know that this is equivalent to what we did on the previous slide.

From the documentation: “It is good practice to use the with keyword when dealing with file objects. The advantage is that the file is properly closed after its suite finishes, even if an exception is raised at some point.”

https://docs.python.org/3/reference/compound_stmts.html#with

In plain English: the with keyword does a bunch of error checking and cleanup for you, automatically.

Reading a File Example

```
# Count the number of words in a text file
in_file = "thesis.txt"
myfile = open(in_file)

num_words = 0
for line_of_text in myfile:
    word_list = line_of_text.split()
    num_words += len(word_list)

myfile.close()

print("Total words in file: ", num_words)
```

Reading a File Multiple Times

You can iterate over a list as many times as you like:

```
mylist = [ 3, 1, 4, 1, 5, 9 ]
for elt in mylist:
    process elt
for elt in mylist:    process is a
    process elt      generic command
```

Iterating over a file uses it up:

```
myfile = open("datafile.dat")
for line_of_text in myfile:
    process line_of_text

for line_of_text in myfile:
    process line_of_text
```

This loop body will never be executed!

How to read a file multiple times?

Solution 1: Read into a list, then iterate over it

```
myfile = open("datafile.dat")
mylines = []
for line_of_text in myfile:
    mylines.append(line_of_text)
use mylines
```

Solution 2: Re-create the file object
(slower, but a better choice if the file does not fit in memory)

```
myfile = open("datafile.dat")
for line_of_text in myfile:
    process line_of_text

myfile = open("datafile.dat")
for line_of_text in myfile:
    process line_of_text
```

Writing files

```
>>> f = open('animals.txt', 'w')  
>>> f.read()
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
IOError: File not open for reading
```

```
>>> f.write('cat\n')  
>>> f.write('dog\n')  
>>> f.write('bird\n')  
>>> f.write('goat\n')
```

Writing files

```
>>> f = open('animals.txt', 'w')  
>>> f.read()
```

Open the file in write mode.
If the file already exists, this
creates it anew, deleting its
old contents.

If I try to read a file in write mode, I get an error.

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
IOError: File not open for reading
```

```
>>> f.write('cat\n')  
>>> f.write('dog\n')  
>>> f.write('bird\n')  
>>> f.write('goat\n')
```

Write to the file. This method returns
the number of characters written to
the file. Note that '\n' counts as a
single character, the new line.

Writing files

```
>>> f = open('animals.txt', 'w')
>>> f.write('cat\n')
>>> f.write('dog\n')
>>> f.write('bird\n')
>>> f.write('goat\n')
>>> f.close()
```

```
>>> f = open('animals.txt', 'r')
>>> for line in f:
...     print(line, end='')
cat
dog
bird
goat
```

Writing files

```
>>> f = open('animals.txt', 'w')
>>> f.write('cat\n')
>>> f.write('dog\n')
>>> f.write('bird\n')
>>> f.write('goat\n')
>>> f.close()
```

Open the file in write mode.
This overwrites the version of the file created in the previous slide.

Each write appends to the end of the file.

When we're done, we close the file. This happens automatically when the program ends, but its good practice to close the file as soon as you're done.

you don't need to use close method if you use "with" keyword

```
>>> f = open('animals.txt', 'r')
>>> for line in f:
...     print(line, end='')
cat
dog
bird
goat
```

Now, when I open the file for reading, I can print out the lines one by one.

The lines of the file already include newlines on the ends, so override Python's default behavior of printing a newline after each line.

>>> f.write('cat\ndog\nbird\ngoat\n')
gives you the same file content

More Examples: Create a file

```
nameHandle = open('characters.txt', 'w')
for i in range(2):
    name = input('Enter name: ')
    nameHandle.write(name + '\n')
nameHandle.close()
```

```
nameHandle = open('characters.txt', 'r')
for line in nameHandle:
    print(line)
nameHandle.close()
```

- If we had typed in the names Rick and Morty, this will print
Rick

Morty
- The extra line between Rick and Morty is there because print starts a new line each time it encounters the '\n' at the end of each line in the file.

More Examples: Overwrite a file

```
nameHandle = open('characters.txt', 'w')
nameHandle.write('Jerry\n')
nameHandle.write('Beth\n')
nameHandle.close()
```

```
nameHandle = open('characters.txt', 'r')
for line in nameHandle:
    print(line[:-1])
nameHandle.close()
```

- It will print
Jerry
Beth
- Notice that
 - we have overwritten the previous contents of the file. Rick and Morty are deleted.
 - print line[:-1] avoids extra newline in the output

More Examples: Append to a file

```
nameHandle = open('characters.txt', 'a')
nameHandle.write('Rick\n')
nameHandle.write('Morty\n')
nameHandle.close()
```

```
nameHandle = open('characters.txt', 'r')
for line in nameHandle:
    print(line[:-1])
nameHandle.close()
```

- It will print
Jerry
Beth
Rick
Morty
- Notice that we can open the file for appending (instead of writing) by using the argument 'a'.

Common functions for accessing files

- `open(fn, 'w')` `fn` is a string representing a file name. Creates a file for writing and returns a file handle.
- `open(fn, 'r')` `fn` is a string representing a file name. Opens an existing file for reading and returns a file handle.
- `open(fn, 'a')` `fn` is a string representing a file name. Opens an existing file for appending and returns a file handle.
- `fn.close()` closes the file associated with the file handle `fn`.

Common functions for accessing files

- `fn.read()` returns a string containing the contents of the file associated with the file handle fn.
- `fn.readline()` returns the next line in the file associated with the file handle fn.
- `fn.readlines()` returns a list each element of which is one line of the file associated with the file handle fn.
- `fn.write(s)` write the string s to the end of the file associated with the file handle fn.
- `fn.writelines(S)` S is a sequence of strings. Writes each element of S to the file associated with the file handle fn.

Formatting Strings

```
>>> x = 23
>>> print('x = %d' % x)
x = 23
```

```
>>> animal = 'unicorn'
>>> print('My pet %s' % animal)
My pet unicorn
```

```
>>> x=2.718; y=1.618
>>> print('%f divided by %f is %f' % (x,y,x/y))
2.718000 divided by 1.618000 is 1.679852
```

```
>>> print('%.3f divided by %.3f is %.8f' % (x,y,x/y))
2.718 divided by 1.618 is 1.67985167
```

Formatting Strings

```
>>> x = 23
>>> print('x = %d' % x)
x = 23
```

Python provides tools for formatting strings. Example: easier way to print an integer as a string.

```
>>> animal = 'unicorn'
>>> print('My pet %s' % animal)
My pet unicorn
```

%d: integer
%s: string
%f: floating point
More information:
<https://docs.python.org/3/library/stdtypes.html#printf-style-string-formatting>

```
>>> x=2.718; y=1.618
>>> print('%f divided by %f is %f' % (x,y,x/y))
2.718000 divided by 1.618000 is 1.67985167
```

Can further control details of formatting, such as number of significant figures in printing floats.

```
>>> print('%.3f divided by %.3f is %.8f' % (x,y,x/y))
2.718 divided by 1.618 is 1.67985167
```

Newer features for similar functionality:
https://docs.python.org/3/reference/lexical_analysis.html#f-strings
<https://docs.python.org/3/library/stdtypes.html#str.format>

Formatting Strings

Note: Number of formatting arguments must match the length of the supplied tuple!

```
>>> x=2.718; y=1.618
```

```
>>> print('%f divided by %f is %f' % (x,y,x/y,1.0))
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: not all arguments converted during string formatting
```

```
>>> x=2.718; y=1.618
```

```
>>> print('%f divided by %f is %f' % (x,y))
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: not enough arguments for format string
```

Formatted Output to a File

```
f = open("students.txt", 'w')
f.write("+-----+-----+-----+\n")
f.write("|First Name      |Last Name      |      GPA|\n")
f.write("+-----+-----+-----+\n")
f.write("|%-15s|%-15s|%10.2f|\n"%(»John", »Smith", 2.75))
f.write("|%-15s|%-15s|%10.2f|\n"%(»Bradley", »Cooper",
3.90))
f.write("+-----+-----+-----+\n")
f.close()
```

```
r = open("students.txt", 'r')
```

```
for l in r:
    print(l[:-1])
```

```
+-----+-----+-----+
|First Name      |Last Name      |      GPA|
+-----+-----+-----+
|John            |Smith          |      2.75|
|Bradley         |Cooper         |      3.90|
+-----+-----+-----+
```

Writing/Reading Binary

```
poem = "Yerin seni çektiği kadar ağırsın\nKanatların çırpındığı kadar  
hafif\nKalbinin attığı kadar canlısın\nGözlerinin uzağı gördüğü kadar genç"
```

```
binary_poem = bytes(poem, encoding="utf-8")
```

```
f = open("binary_poem", "wb")  
f.write(binary_poem)  
f.close()
```

```
$ ls  
binary_poem      students.txt  
$ more binary_poem  
"binary_poem" may be a binary file.  See it anyway? █
```

```
f = open("binary_poem", "rb")  
data = f.read() # read whole file  
print(data)  
f.close()
```

```
b'Yerin seni \xc3\xa7ekti\xc4\x9fi kadar  
a\xc4\x9f\xc4\xb1rs\xc4\xb1n\nKanatlar\xc4\xb1n  
\xc3\xa7\xc4\xb1rp\xc4\xb1nd\xc4\xb1\xc4\x9f\xc4\xb1  
kadar hafif\nKalbinin att\xc4\xb1\xc4\x9f\xc4\xb1  
kadar canl\xc4\xb1s\xc4\xb1n\nG\xc3\xb6zlerinin  
uza\xc4\x9f\xc4\xb1  
g\xc3\xb6rd\xc3\xbcc\xc4\x9f\xc3\xbcc kadar  
gen\xc3\xa7'
```

```
f = open("binary_poem", "rb")  
chunk = 20 # read as chunks  
while True:  
    data = f.read(chunk)  
    if not data:  
        break  
    print(data)  
f.close()
```

```
b'Yerin seni \xc3\xa7ekti\xc4\x9fi'  
b' kadar a\xc4\x9f\xc4\xb1rs\xc4\xb1n\nKa'  
b'natlar\xc4\xb1n \xc3\xa7\xc4\xb1rp\xc4\xb1nd'  
b'\xc4\xb1\xc4\x9f\xc4\xb1 kadar hafif\nK'  
b'albinin att\xc4\xb1\xc4\x9f\xc4\xb1 ka'  
b'dar canl\xc4\xb1s\xc4\xb1n\nG\xc3\xb6zl'  
b'lerinin uza\xc4\x9f\xc4\xb1 g\xc3\xb6rd'  
b'\xc3\xbcc\xc4\x9f\xc3\xbcc kadar gen\xc3\xa7'
```

Lecture Overview

- File I/O
 - Basics of Files
 - File Access Modes
 - Sequential Access
 - Random Access

seek() and tell() Functions

- So far we have read and write files sequentially.
- It is also possible to read and write at specific locations in a file.
- To achieve this the file object provides following two methods:

fileObject.seek(offset[, whence]) : Sets the file's current position.

offset – This is the position of the read/write pointer within the file.

whence – This is optional and defaults to 0 which means absolute file positioning, other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

fileObject.tell() : Returns an integer giving the file object's current position in the file represented as number of bytes from the beginning of the file when in binary mode and an opaque number when in text mode.

Random Access: Example

```
f = open("digits.txt", "w")
f.write("0\n1\n2\n3\n4\n5\n6\n7\n8\n9\n")
f.close()
```

```
f = open("digits.txt", "r")
print(f.read())
f.close()
```

This is how it
looks when
printed.

0
1
2
3
4
5
6
7
8
9

This is how it
looks on the
file.

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	0	\n	1	\n	2	\n	3	\n	4	\n	5	\n	6	\n	7	\n	8	\n	9	\n

Random Access: Example (cont'd)

The first byte is positioned
when the file is opened.

```
f = open("digits.txt", "r")  
l = f.readline()  
print(f.tell())
```

```
f.seek(10)  
print(f.readline())  
print(f.tell())  
f.close()
```

```
f = open("digits.txt", "r+")  
f.seek(10)  
f.write("*")  
print(f.tell())
```

```
f.seek(10)  
print(f.readline())  
f.close()
```

Position

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	\n	1	\n	2	\n	3	\n	4	\n	5	\n	6	\n	7	\n	8	\n	9	\n

Random Access: Example (cont'd)

First line is read (new line character is included).

File is positioned at line #2.

```
f = open("digits.txt", "r")  
l = f.readline()  
print(f.tell())
```

2

```
f.seek(10)  
print(f.readline())  
print(f.tell())  
f.close()
```

```
f = open("digits.txt", "r+")  
f.seek(10)  
f.write("*")  
print(f.tell())
```

```
f.seek(10)  
print(f.readline())  
f.close()
```

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	0	\n	1	\n	2	\n	3	\n	4	\n	5	\n	6	\n	7	\n	8	\n	9	\n

Random Access: Example (cont'd)

File is positioned at line #5, and then read.

File is positioned at line #6.

```
f = open("digits.txt", "r")
l = f.readline()
print(f.tell())
```

2

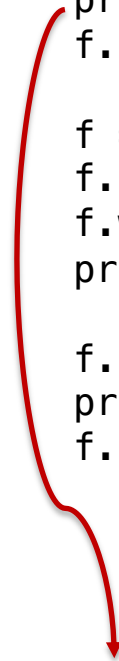
```
f.seek(10)
print(f.readline())
print(f.tell())
f.close()
```

5

12

```
f = open("digits.txt", "r+")
f.seek(10)
f.write("*")
print(f.tell())
```

```
f.seek(10)
print(f.readline())
f.close()
```



Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	0	\n	1	\n	2	\n	3	\n	4	\n	5	\n	6	\n	7	\n	8	\n	9	\n

Random Access: Example (cont'd)

File is positioned at line #5 again.

File is positioned at line #6.

```
f = open("digits.txt", "r")  
l = f.readline()  
print(f.tell())
```

2

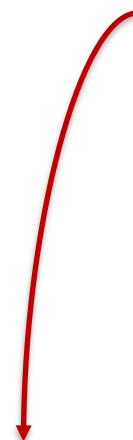
```
f.seek(10)  
print(f.readline())  
print(f.tell())  
f.close()
```

5

12

```
f = open("digits.txt", "r+")  
f.seek(10)  
f.write("*")  
print(f.tell())
```

```
f.seek(10)  
print(f.readline())  
f.close()
```



Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	0	\n	1	\n	2	\n	3	\n	4	\n	5	\n	6	\n	7	\n	8	\n	9	\n

Random Access: Example (cont'd)

```
f = open("digits.txt", "r")  
l = f.readline()  
print(f.tell())
```

2

```
f.seek(10)  
print(f.readline())  
print(f.tell())  
f.close()
```

5

12

```
f = open("digits.txt", "r+")  
f.seek(10)  
f.write("*")  
print(f.tell())
```

11

```
f.seek(10)  
print(f.readline())  
f.close()
```



Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	0	\n	1	\n	2	\n	3	\n	4	\n	*	\n	6	\n	7	\n	8	\n	9	\n

Random Access: Example (cont'd)

```
f = open("digits.txt", "r")  
l = f.readline()  
print(f.tell())
```

2

```
f.seek(10)  
print(f.readline())  
print(f.tell())  
f.close()
```

5

12

```
f = open("digits.txt", "r+")  
f.seek(10)  
f.write("*")  
print(f.tell())
```

11

```
f.seek(10)  
print(f.readline())  
f.close()
```

*



Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	0	\n	1	\n	2	\n	3	\n	4	\n	*	\n	6	\n	7	\n	8	\n	9	\n

Example: Fixed Length Records

```
record_length = 40          # Length of a student record
search_term = »Ismail"      # student to be looked up in the file
```

```
f = open("grades.txt", "w")
f.write("%-15s%-15s%10.2f"%(»Emre", "Sefer", 2.75))
f.write("%-15s%-15s%10.2f"%(»Ismail", »Akturk", 3.90))
f.write("%-15s%-15s%10.2f"%(»Nitel", "Muhtaroglu", 3.50))
f.close()
```

Position in
the file



Content of
the file



0	39	40	79	80	119
Emre	Sefer	2.75	Ismail	Akturk	3.90
Nitel	Muhtaroglu	3.50			

File: grades.txt

```
index = {»Emre":0, »Ismail":1, "Nitel":2} # index students
```

```
f = open("grades.txt", "r")
f.seek(index[search_term] * record_length)
print(f.read(record_length))
f.close()
```

Output:

Ismail Akturk 3.90

Next time... Testing, debugging, exceptions

Exceptions



```
try:
```

```
.....
```

```
.....
```

```
except:
```

```
.....
```

```
finally:
```

```
.....
```



Debugging

