ITP 115

Files

Outline

- File overview
- Read from text files
- Write to text files

What Happens to Variables?

- While your program is running, all the variables are stored in the computer's memory (RAM)
- When your program stops, all those variables are destroyed
  - What if you want that data later?
- Files allow for permanent storage of information

What is a File?

- A collection of information
  - stored in units of data called *bytes*
- Files reside on your computer's hard drive, your phone's memory, etc.
  - Files exist after your program stops
- Files can be transferred over email, wifi, Bluetooth, etc.

Kinds of Files

- Files are either stored as **Text** or **Binary**

- Text files store data in human-readable formats
  - Ex: Simple text files (.txt) or web pages (.html)

- Binary files data in computer-readable formats
  - Ex: pictures (.jpg), music (.mp3), or Word doc (.docx)

How Will We Use Files?

- To save data when the program stops
- To share information
- To write programs that use multiple data files

Why Text Files Specifically?

- Great for storing simple information like strings (or ints we can convert to strings)
- They are cross-platform
- They are easy to use
  - Most operating systems come with basic tools to view and edit them

Reading from a File

Three Step Process

1. Open the file for **reading**
2. Read from the file
3. Close the file

Reading Files

- **Reading** is the process of getting data from a file that is on your computer

- To access a file, we need to create connection between our Python program and the file
  - Think: *a pipe*

1. Opening a File for Reading

- Use the built-in function **`open()`**

```
fileIn = open("words.txt", "r")
```

filename

file access mode

- Two parameters
  - Name of the file (in current directory)
  - File access mode (**read mode**)

- Specify you want to **R**ead from the file (input)

1. Opening a File for Reading

```
fileIn = open("words.txt", "r")
```

↑

file object

- Returns a **file object** that you use to read the file
- Think of file object a *"pipe"* that connects to the text file so that you can read from the file

USC School of Engineering

University of Southern California

1. Opening a File for Reading

program.py

```
def main():
   fileIn = open("words.txt", "r")
```

words.txt

```
Tommy Trojan
Traveler
George Tirebiter
```

1. Opening a File for Reading

program.py

```
def main():
   fileIn = open("words.txt", "r")
```

words.txt

```
Tommy Trojan
Traveler
George Tirebiter
```

2. Reading from a File

```
fileIn = open("words.txt", "r")
for line in fileIn:
    print(line)
```

- One elegant solution to move through all of the lines of a text file is to use a `for` loop

- Each time through the `for` loop, one entire line is read from the file (everything up to the next `\n`)

Reading Files

program.py

```
def main():
  fileIn = open("words.txt", "r")
  for line in fileIn:
    print(line)
```

words.txt

```
Tommy Trojan
Traveler
George Tirebiter
```

2. Reading from a File

```
for line in fileIn:
      print(line)
```

words.txt

```
Tommy Trojan
Traveler
George Tirebiter
```

line

Print output

2. Reading from a File

```
for line in fileIn:
    print(line)
```

1st Iteration

words.txt

```
Tommy Trojan
Traveler
George Tirebiter
```

line

```
Tommy Trojan
```

Print output

2. Reading from a File

```
for line in fileIn:
    print(line)
```

1st Iteration

words.txt

```
Tommy Trojan
Traveler
George Tirebiter
```

line `Tommy Trojan`

Print output

```
Tommy Trojan
```

2. Reading from a File

```
for line in fileIn:
    print(line)
```

2nd Iteration

words.txt

```
Tommy Trojan
Traveler
George Tirebiter
```

```
line  Traveler
```

Print output

```
Tommy Trojan
```

2. Reading from a File

```
for line in fileIn:
    print(line)
```

2nd Iteration

words.txt

```
Tommy Trojan
Traveler
George Tirebiter
```

line
```
Traveler
```

Print output

```
Tommy Trojan

Traveler
```

2. Reading from a File

```
for line in fileIn:
    print(line)
```

3rd Iteration

words.txt

```
Tommy Trojan
Traveler
George Tirebiter
```

line `George Tirebiter`

Print output

```
Tommy Trojan

Traveler
```

2. Reading from a File

```
for line in fileIn:
    print(line)
```

3rd Iteration

words.txt

```
Tommy Trojan
Traveler
George Tirebiter
```

line `George Tirebiter`

Print output

```
Tommy Trojan

Traveler

George Tirebiter
```

Something is Not Quite Right…

words.txt

```
Tommy Trojan
Traveler
George Tirebiter
```

Print output

```
Tommy Trojan

Traveler

George Tirebiter
```

- What's happening?

Something is Not Quite Right…

words.txt

```
Tommy Trojan\n
Traveler\n
George Tirebiter\n
```

Print output

```
Tommy Trojan

Traveler

George Tirebiter
```

- The text file has "invisible" newline characters (**\n**)
- These newlines are at the end of every line of **words.txt**

Something is Not Quite Right…

words.txt

```
Tommy Trojan\n
Traveler\n
George Tirebiter\n
```

Print output

```
Tommy Trojan

Traveler

George Tirebiter
```

- The **print** function also adds a newline
- This means we get double newlines

Something is Not Quite Right…

words.txt

```
Tommy Trojan\n
Traveler\n
George Tirebiter\n
```

Print output

```
Tommy Trojan

Traveler

George Tirebiter
```

- Solution:
  Remove the newline when we read from the file

`someString.strip()`

- Remove <u>any</u> white space from the **beginning** <u>and</u> the **end** of the `someString`

- Returns the edited string

- Whitespace is
  - A space
  - A tab (`\t`)
  - A newline (`\n`)

someString.strip()

- Example

```
msg = "   French\nPress\n\n"
print(msg, "coffee")
```

Print output

```
   French
Press


coffee
```

```
msg = "   French\nPress\n\n"
msg = msg.strip()
print(msg, "coffee")
```

```
French
Press coffee
```

`someString.strip()`

```
msg = "   French\nPress\n\n"
msg = msg.strip()
print(msg, "coffee")
```

Print output

```
French
Press coffee
```

- Notice that `strip` does not remove the `\n` in the middle of the string

2. Revised Reading from a File

- Instead of

```
for line in fileIn:
    # loop body code
```

- We will use

```
for line in fileIn:
    line = line.strip()
    # loop body code
```

2. Revised Reading from a File

```
for line in fileIn:
    line = line.strip()
    print(line)
```

words.txt

```
Tommy Trojan\n
Traveler\n
George Tirebiter\n
```

Print output

```
Tommy Trojan
Traveler
George Tirebiter
```

3. Close the File

- AFTER you are done reading the file, close it

```
fileIn = open("words.txt", "r")
for line in fileIn:
   word = line.strip()
   print(word)
fileIn.close()
```

- This closes the "pipe" and prevents any possible corruption of the file

One More Thing…

- What kind of variable is `line`?

```
for line in fileIn:
    line = line.strip()
    print(line)
```

One More Thing…

- What kind of variable is `line`?

```
for line in fileIn:
    line = line.strip()
    print(line)
```

- `line` will always be a **string**

One More Thing…

- What if we have a file of **ints**?
  - Need to convert `line` to **int** (just like with `input()` function)

numbers.txt

```
3\n
-8\n
15\n
```

File Examples

```
for line in fileIn:
  line = int(line.strip())
  print(2*line)
```

numbers.txt

```
3\n
-8\n
15\n
```

Print output

```
6
-16
30
```

- *End lecture*

Writing to a File

Three Step Process

1. Open the file for **writing**
2. Write to the file
3. Close the file

Writing Files

- **Writing** is the process of putting data into a file that is on your computer

- To access a file, we need to create connection between our Python program and the file
  - Think: *a pipe*

1. Opening a File for Writing

- Use the built-in function **open()**

```
fileOut = open("words.txt", "w")
```

filename ***file access mode***

- Two parameters
  - Name of the file (in current directory)
  - File access mode (**write mode**)

- Specify you want to **W**riting from the file (output)

1. Opening a File for Writing

```
fileOut = open("words.txt", "w")
```

↑

file object

- Returns a **file object** that you use to write to the file
- Think of file object a *"pipe"* that connects to the text file so that you can write to the file

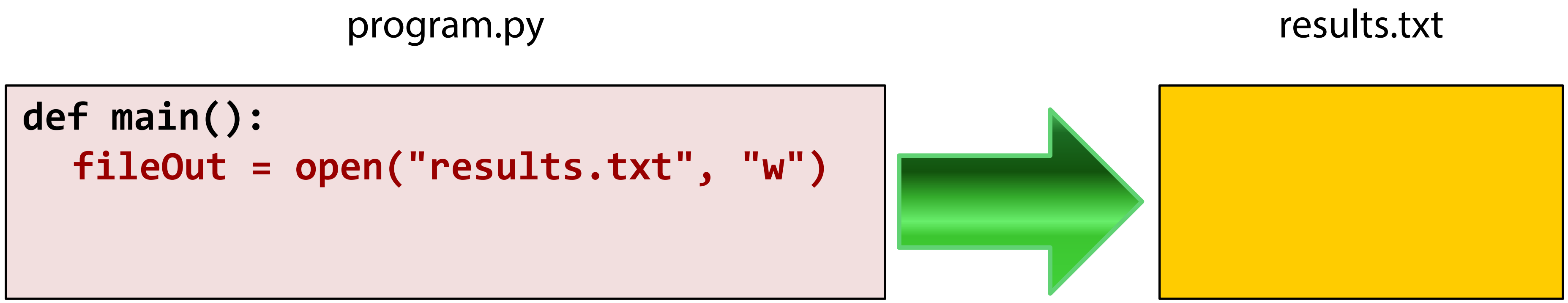
1. Opening a File for Writing

program.py

```
def main():
  fileOut = open("results.txt", "w")
```

results.txt

1. Opening a File for Writing



```
def main():
   fileOut = open("results.txt", "w")
```

2. Writing To a Text File

- Use the `print` function with the `file` argument
  - Similar to the `end` argument we used earlier

```
fileOut = open("results.txt", "w")
print("Hello World", file=fileOut)
```

- This code write "`Hello World`" to *results.txt*

2. Writing To a Text File

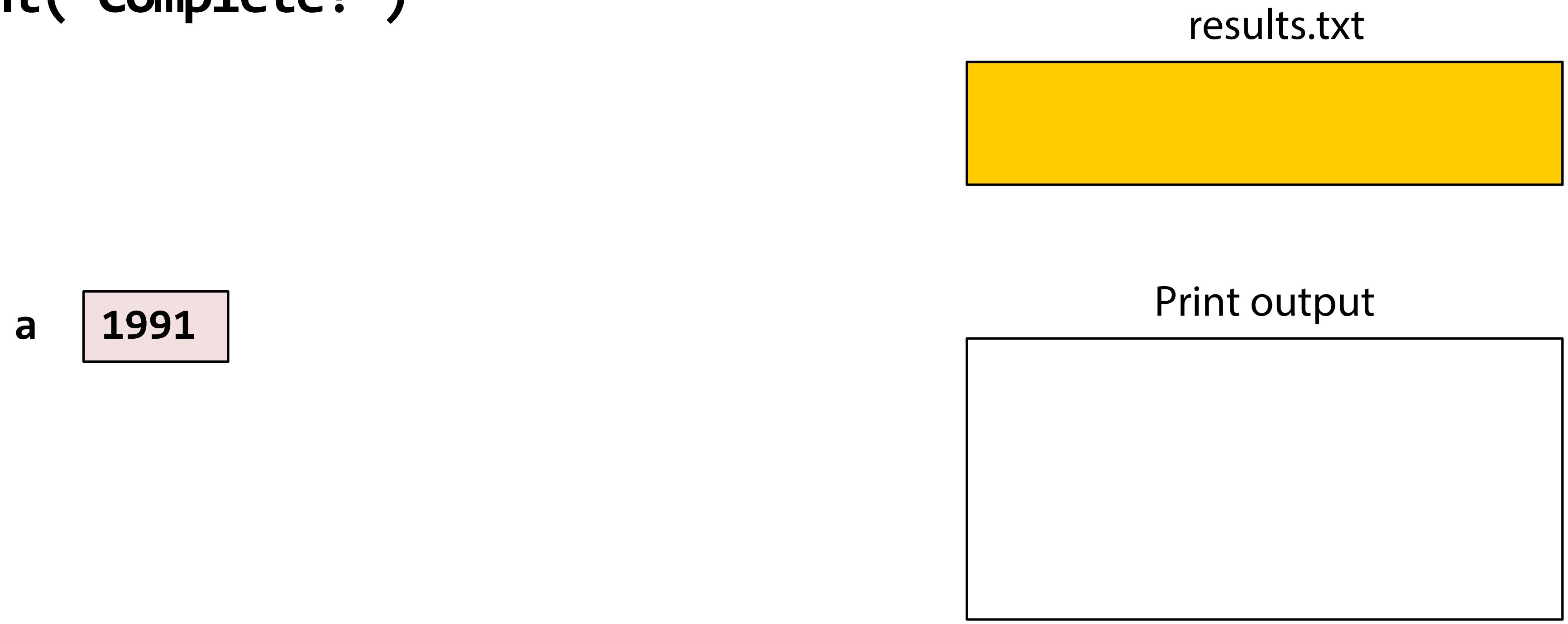
```
a = 1991
print("Hello World", file=fileOut)
print("Python was born in", a, file=fileOut)
print("Complete!")
```

results.txt

Print output

2. Writing To a Text File

```
a = 1991
print("Hello World", file=fileOut)
print("Python was born in", a, file=fileOut)
print("Complete!")
```



2. Writing To a Text File

```
a = 1991
print("Hello World", file=fileOut)
print("Python was born in", a, file=fileOut)
print("Complete!")
```

results.txt

```
Hello World
```

a
```
1991
```

Print output

2. Writing To a Text File

```
a = 1991
print("Hello World", file=fileOut)
print("Python was born in", a, file=fileOut)
print("Complete!")
```

results.txt

```
Hello World
Python was born in 1991
```

```
a  1991
```

Print output

2. Writing To a Text File

```
a = 1991
print("Hello World", file=fileOut)
print("Python was born in", a, file=fileOut)
print("Complete!")
```

results.txt

```
Hello World
Python was born in 1991
```

```
a  1991
```

Print output

```
Complete!
```

3. Close the File

- AFTER you are done writing to the file, close it

```
fileOut = open("results.txt", "w")
print("Hello World", file=fileOut)
fileOut.close()
```

- This closes the "pipe" and prevents any possible corruption of the file

Text File Access Modes

Mode	Description
`"r"`	**Read** from a file. If the file doesn't exist, Python will generate an error.
`"w"`	**Write** to a file. If the file exists, its contents are overwritten. If the file doesn't exist, it's created.
`"a"`	**Append** a file. If the file exists, new data is appended to it. If the file doesn't exist, it's created.

Reference only

More Text File Access Modes

Mode	Description
`"r+"`	**Read** from and **write** to a file. If the file doesn't exist, Python will generate an error.
`"w+"`	**Write** to and **read** from a file. If the file exists, its contents are overwritten. If the file doesn't exist, it's created.
`"a+"`	**Append** and **read** from a file. If the file exists, new data is appended to it. If the file doesn't exist, it's created.

Comma-Separated Value (CSV) Files

- CSV files are often used to exchange data
- CSV files are text-files that can be used to represent the same data as a spreadsheet
- CSV files are convenient because they can be read by any program, platform, etc.

CSV Format

- Each **row** represents one **line** in a table
- **Commas** separate each **column**
- The first line often represent the headers

CSV Example

DEPT	COURSE_NUMBER	SEMESTER	NUMBER_OF_STUDENTS
ITP	115	Spring17	30
BUAD	101	Spring17	40
ITP	310	Spring17	35

CSV Example

DEPT	COURSE_NUMBER	SEMESTER	NUMBER_OF_STUDENTS
ITP	115	Spring17	30
BUAD	101	Spring17	40
ITP	310	Spring17	35

`class.csv`

```
DEPT,COURSE_NUMBER,SEMESTER,NUMBER_OF_STUDENTS
```

CSV Example

DEPT	COURSE_NUMBER	SEMESTER	NUMBER_OF_STUDENTS
ITP	115	Spring17	30
BUAD	101	Spring17	40
ITP	310	Spring17	35

class.csv

```
DEPT,COURSE_NUMBER,SEMESTER,NUMBER_OF_STUDENTS

ITP,115,SPRING17,30
```

CSV Example

DEPT	COURSE_NUMBER	SEMESTER	NUMBER_OF_STUDENTS
ITP	115	Spring17	30
BUAD	101	Spring17	40
ITP	310	Spring17	35

class.csv

```
DEPT,COURSE_NUMBER,SEMESTER,NUMBER_OF_STUDENTS
ITP,115,SPRING17,30
BUAD,101,SPRING17,40
```

CSV Example

DEPT	COURSE_NUMBER	SEMESTER	NUMBER_OF_STUDENTS
ITP	115	Spring17	30
BUAD	101	Spring17	40
ITP	310	Spring17	35

class.csv

```
DEPT,COURSE_NUMBER,SEMESTER,NUMBER_OF_STUDENTS
ITP,115,SPRING17,30
BUAD,101,SPRING17,40
ITP,310,SPRING17,35
```

Processing CSV Files

```
DEPT,COURSE_NUMBER,SEMESTER,NUMBER_OF_STUDENTS
ITP,115,SPRING17,30
BUAD,101,SPRING17,40
ITP,310,SPRING17,35
```

- How do we process CSV files?

Processing CSV Files

```
DEPT,COURSE_NUMBER,SEMESTER,NUMBER_OF_STUDENTS
ITP,115,SPRING17,30
BUAD,101,SPRING17,40
ITP,310,SPRING17,35
```

- How do we process CSV files?
  - Open the file and read line by line
  - Use the **`split()`** function
  - Close the file

Split

- The split function breaks a large string down into smaller strings using a defined separator
  - If no separator is defined, then whitespace will be used by default
- For CSV files, what should we use as the delimiter?

Split

- The split function breaks a large string down into smaller strings using a defined separator
  - If no separator is defined, then whitespace will be used by default
- For CSV files, what should we use as the delimiter?

a comma

","

someString.split(",")

- Example

```
line = "red,green,blue"
colorList = line.split(",")
print(colorList)

print(colorList[0])
print(colorList[1])
print(colorList[2])
```

Print output

```
['red', 'green', 'blue']
```

```
red
green
blue
```

Header Row

- Most CVS files have a header row to label each column
- We don't want to put those labels into our data
- To read the first line, use a file object method called **`readline()`**
  - It reads all of the characters to the end of the line

DEPT	COURSE_NUMBER	SEMESTER	NUMBER_OF_STUDENTS
ITP	115	Spring17	30
BUAD	101	Spring17	40
ITP	310	Spring17	35

fileobj.readline()

- Example

```
fileIn = open("data.csv", "r")
# Skip header row
fileIn.readline()
for line in fileIn:
    line = line.strip()
    dataList = line.split(",")
    numStr = dataList[3]
    num = int(numStr)
    print("Number of students is", num)
fileIn.close()
```

ALTERNATE FILE METHODS

Reference only

Alternate File Methods

Method	Description
`read([size])`	Reads **`size`** characters from a text file and returns them as a string. If **`size`** is not specified, the method returns all of the characters from the current position to the end of the file.
`readline([size])`	Reads **`size`** characters from the current line in a text file and returns them as a string. If **`size`** is not specified, the method returns all of the characters from the current position to the end of the line.
`readlines()`	Reads all of the lines in a text file and returns them as elements in a list.
`write(output)`	Writes the string **`output`** to a text file.
`writelines(output)`	Writes the strings in the list **`output`** to a text file.

Reading Individual Characters

- Use the **`read(n)`** function to read the next ***n*** number of characters
- Python remembers where it last read, and each subsequent **`read(n)`** begins where the last ended
- To start back at the beginning of a file, close and open it
- If you don't specify a number, Python returns the entire file as a string
  - Only good for small files

Reading Characters from a Line

- Use the **`readline(`*`n`*`)`** method where ***n*** is the number of characters you want to read from the current line
- The method returns the characters as a string
- Once you read all of the characters of a line, the next line becomes the current line
- If you don't pass a number, the method returns the entire line
- **`readline()`** reads characters from the current line only, while **`read()`** reads characters from the entire file

Reference only

Reading the Entire File at Once

`readlines()`

- Read the **entire** text file into a list
- Each line of the file becomes a separate string element in the list

Writing Individual Strings To a Text File

Reference only

- Writing strings to a text file
  - Use the **`write(`*`string`*`)`** method which writes a string to a text file
  - **`write()`** does not automatically insert a newline character at the end of a string
  - You have to put newlines in where you want them (use **`\n`**)

Reference only

Writing a List of Strings To a Text File

- Writing a list of strings to a text file
  - Use the `writelines(someList)` method
  - `someList` is a list of strings