BIOS6642

Files

Fuyong Xing

Department of Biostatistics and Informatics

Colorado School of Public Health

University of Colorado Anschutz Medical Campus

Outline

- **Reading and writing**
- The *with* statement
- CSV files
- Filenames and paths

Reading and writing

- We often need to analyze data stored as files in secondary storage such as hard disk drives.
- The *open* function in Python opens a file and returns a file object. If the file can not be opened, an exception is raised.

```
f =  open('myfile.txt', 'r')
```

File name

The mode in which the file is opened. Here 'r' means open for reading.

Reading and writing

- The modes for the *open* function

Character	Meaning
`'r'`	open for reading (default)
`'w'`	open for writing, truncating the file first
`'x'`	open for exclusive creation, failing if the file already exists
`'a'`	open for writing, appending to the end of the file if it exists
`'b'`	binary mode
`'t'`	text mode (default)
`'+'`	open for updating (reading and writing)

https://docs.python.org/3/library/functions.html#open

Reading and writing

- Object methods to read data from *myfile.txt*
  - The *read* method

```
>>> f = open('myfile.txt', 'r')
>>> f.read()
'Bios\n6642\nPython\n'
```

Reading and writing

- Object methods to read data from *myfile.txt*
  - The *readline* method

```
>>> f = open('myfile.txt', 'r')
>>> f.readline()
'Bios\n'
>>> f.readline()
'6642\n'
>>> f.readline()
'Python\n'
```

Reading and writing

- Object methods to read data from *myfile.txt*
  - The *readlines* method

```
>>> f = open('myfile.txt', 'r')
>>> f.readlines()
['Bios\n', '6642\n', 'Python\n']
```

Reading and writing

- The *write* method writes the contents of string to the file (e.g., *myfile_output.txt*), returning the number of characters written.


```
>>> f = open('myfile_output.txt', 'w')
>>> f.write('Introduction\n')
13
>>> f.write('to\n')
3
>>> f.write('Python\n')
7
>>> f.write('Programming\n')
12
```

Reading and writing

- Print each line of a file (e.g., *data.dat*) with a *for* loop

```
f = open('data.dat')        # f is a file object
for line in f:              # Read each line as text
    print(line.strip())     # Remove trailing newline character
f.close()                   # Close the file
```

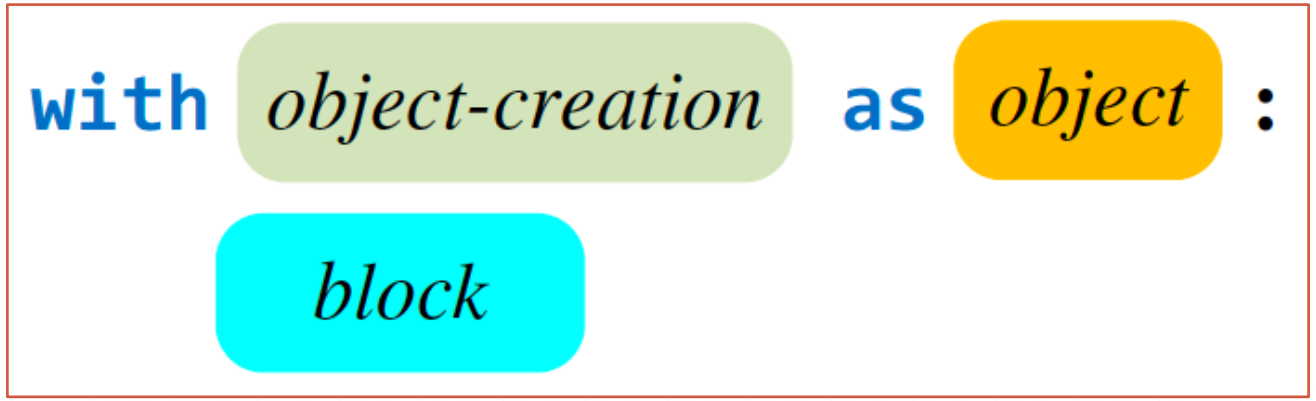
When the program completes the file processing, it should call the *close* method to close the file.

Outline

- Reading and writing
- **The *with* statement**
- CSV files
- Filenames and paths

The *with* statement

- Use the *with* statement to automatically close the file



- The reserved word *with* begins the *with* statement.
- The expression *object-creation* attempts to create and return an object.
- The reserved word *as* binds the object created by the object-creation expression to a variable.
- *object* is bound to the object created by the object-creation expression.
- *block* contains the code that uses the object bound to *object*.

The *with* statement

- Example: enter numbers from the keyboard and save them to a file

Codes

```
"""
Uses Python's file class to store data to and retrieve data from
a text file.
"""
def load_data(filename):
    """ Print the elements stored in the text file named filename. """
    # Open file to read
    with open(filename) as f: # f is a file object
        for line in f: # Read each line as text
            print(int(line)) # Convert to integer and append to the list

def store_data(filename):
    """ Allows the user to store data to the text file named filename. """
    with open(filename, 'w') as f: # f is a file object
        number = 0
        while number != 999: # Loop until user provides magic number
            number = int(input('Please enter number (999 quits):'))
            if number != 999:
                f.write(str(number) + '\n') # Convert integer to string to save
            else:
                break # Exit loop
```

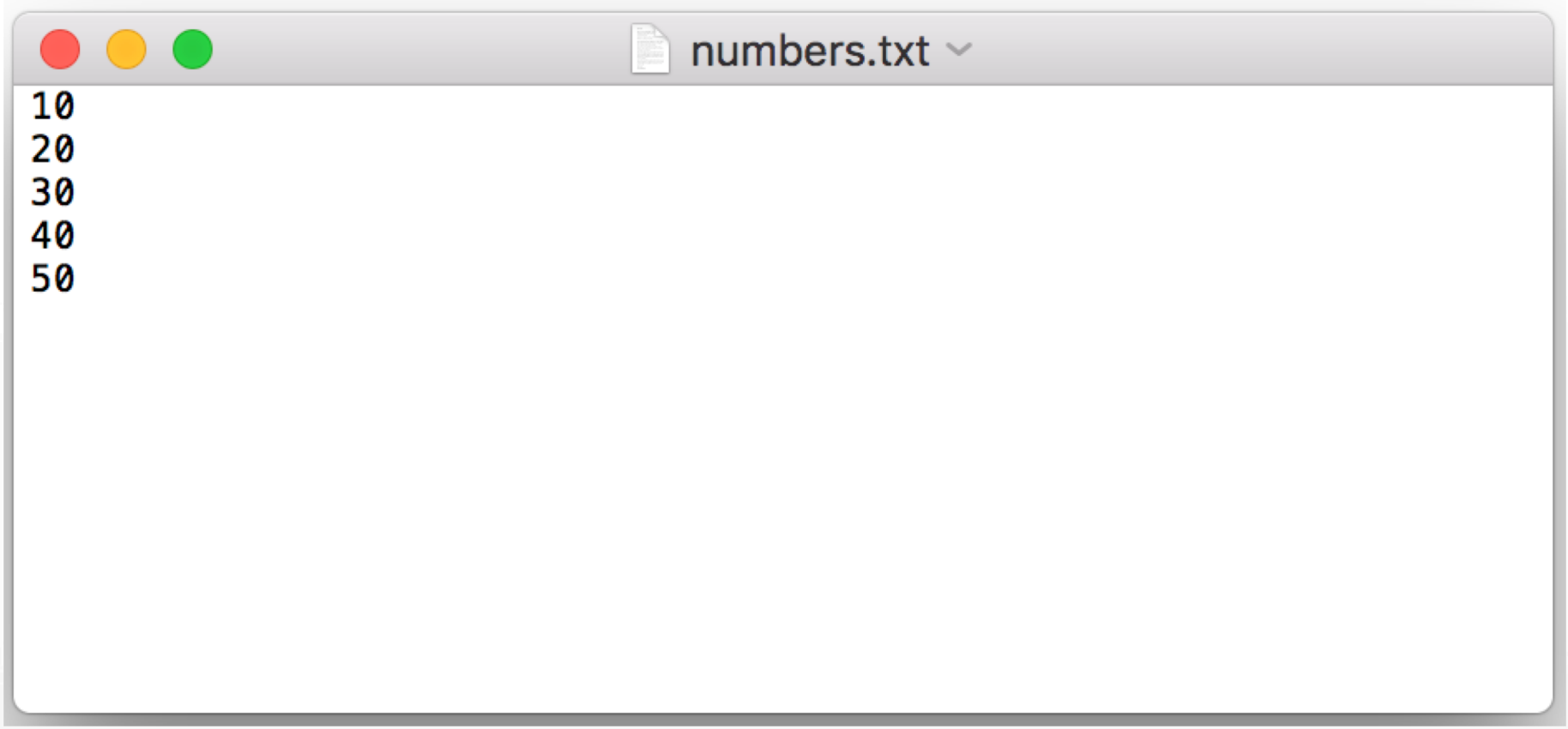
```
def main():
    """ Interactive function that allows the user to
        create or consume files of numbers. """
    done = False
    while not done:
        cmd = input('S)ave L)oad Q)uit: ')
        if cmd == 'S' or cmd == 's':
            store_data(input('Enter file name:'))
        elif cmd == 'L' or cmd == 'l':
            load_data(input('Enter filename:'))
        elif cmd == 'Q' or cmd == 'q':
            done = True

if __name__ == '__main__':
    main()
```

```
S)ave L)oad Q)uit: S
Enter file name:numbers.txt
Please enter number (999 quits):10
Please enter number (999 quits):20
Please enter number (999 quits):30
Please enter number (999 quits):40
Please enter number (999 quits):50
Please enter number (999 quits):999
S)ave L)oad Q)uit: q
```

The *with* statement

- Example: enter numbers from the keyboard and save them to a file

Output



The *with* statement

- Example: enter numbers from the keyboard and save them to a file

Retrieve the numbers in the file

```
S)ave L)oad Q)uit: l
Enter filename:numbers.txt
10
20
30
40
50
S)ave L)oad Q)uit: q
```

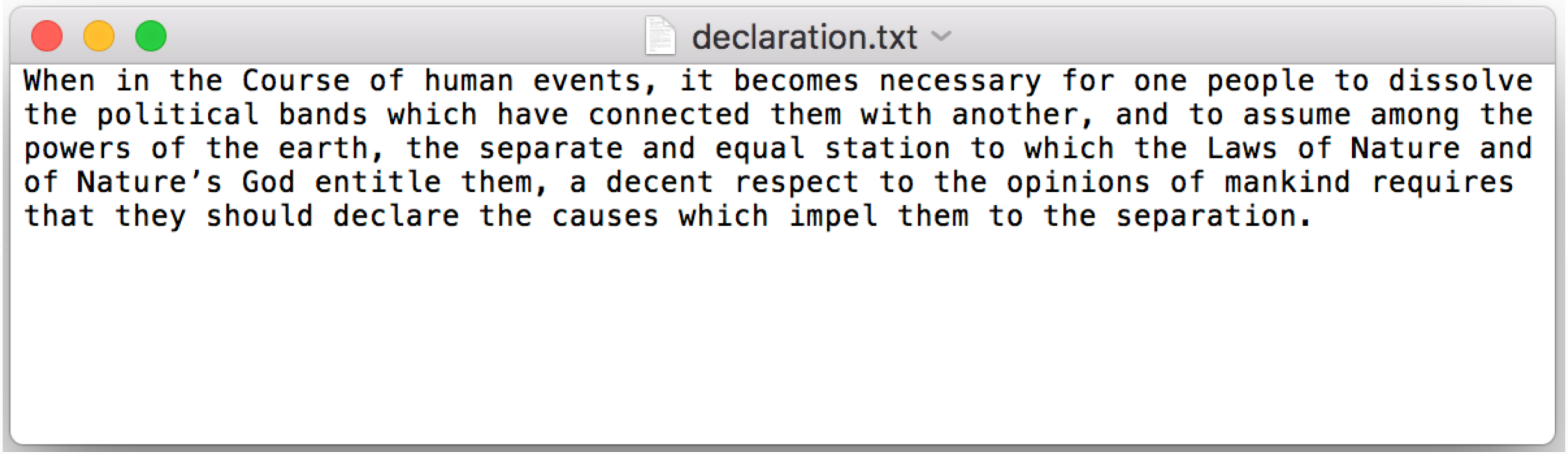
The *with* statement

- Example: capitalize the text in a file and save the results into a new file

```
def capitalize(filename):
    """ Creates a new file with the prefix 'upper_'
        added to the name of the original file.
        All the alphabetic characters in the new
        are capitalized. This function does not
        disturb the contents of the original file. """
    with open(filename, 'r') as infile:
        with open('upper_' + filename, 'w') as outfile:
            for line in infile:
                line = line.strip().upper()
                print(line, file=outfile)

capitalize('declaration.txt')
```

The *with* statement

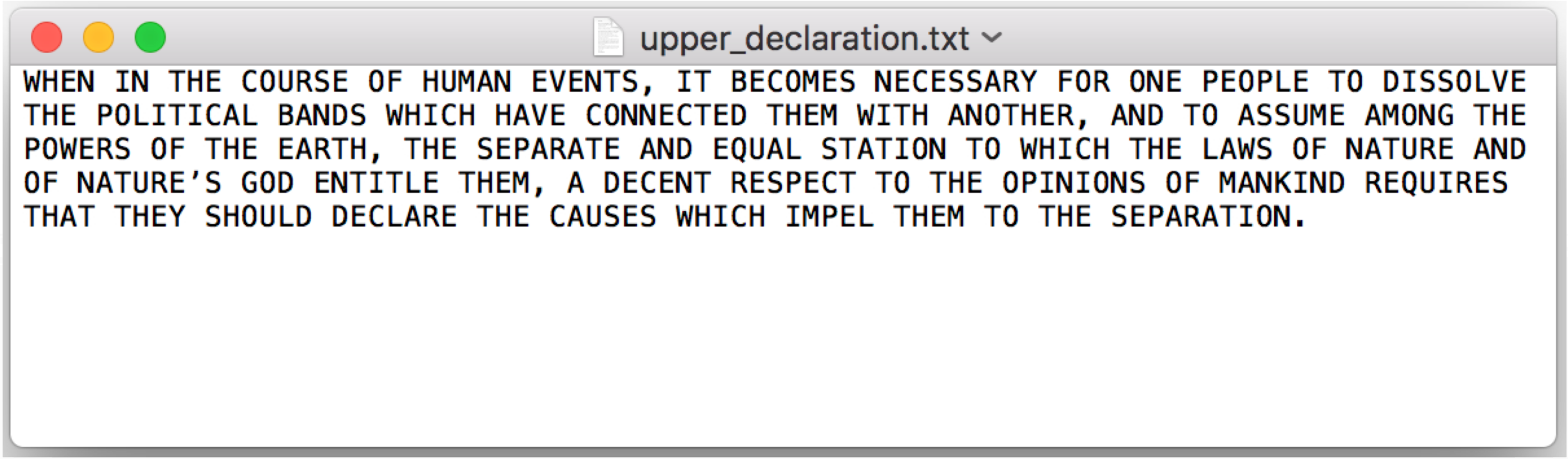
- Example: capitalize the text in a file and save the results into a new file



Input

The *with* statement

- Example: capitalize the text in a file and save the results into a new file



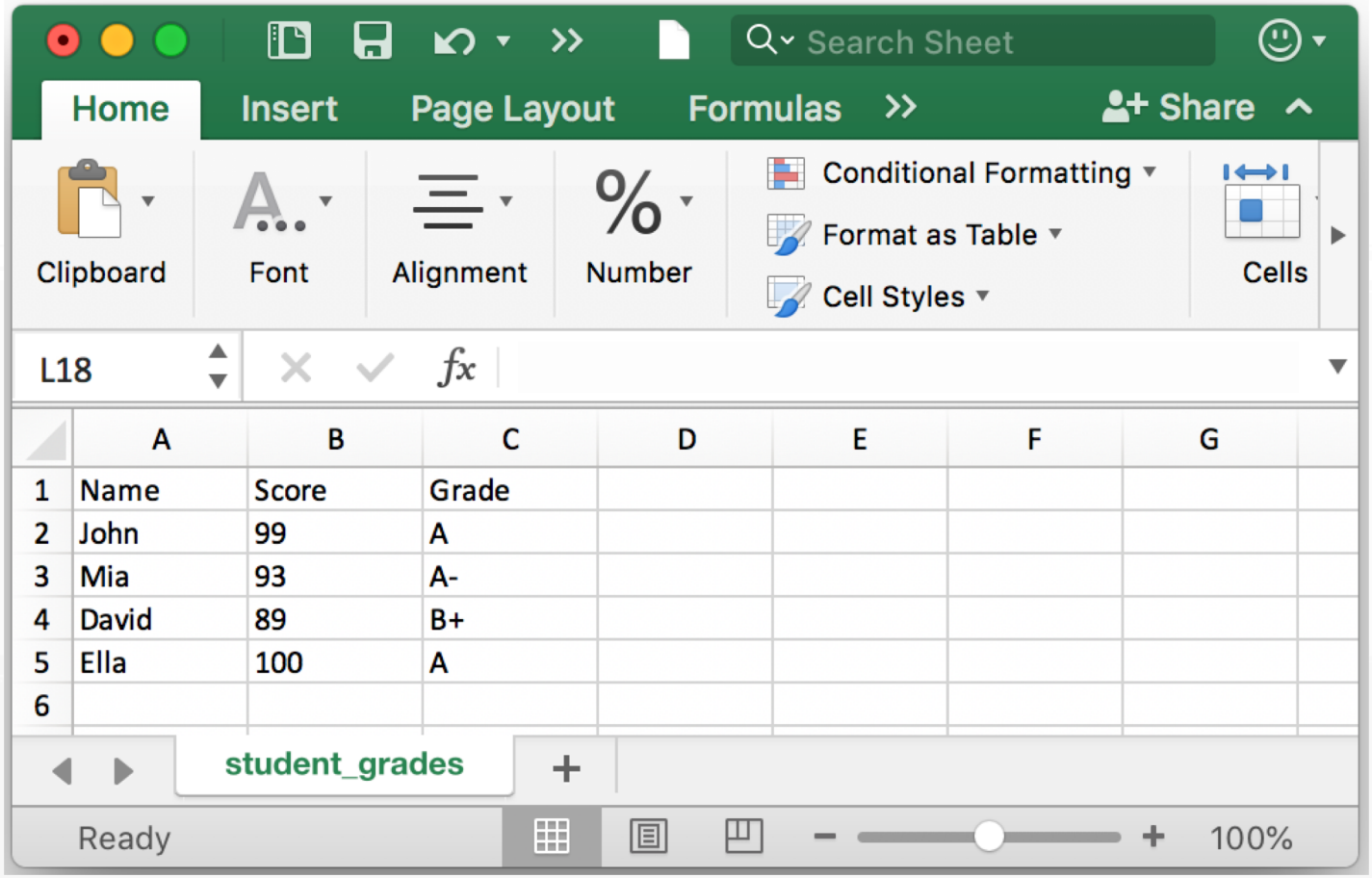
Output

Outline

- Reading and writing
- The *with* statement
- **CSV files**
- Filenames and paths

CSV files

- A comma-separated values (CSV) file can be easily imported into other programs like excel, Google sheet, or a statistics package (e.g., R).
- A CSV file typically has a header as the first line that contains column names.



CSV files

- We can read data in CSV files the same way we have for the text file.

Codes

```
stu_grd = open("student_grades.csv", 'r')
lines = stu_grd.readlines()
header = lines[0]
names = header.strip().split(',')
print("{0:8} {1:8} {2:8}".format(names[0], names[1], names[2]))
for row in lines[1:]:
    values = row.strip().split(',')
    if values[2] == "A":
        print("{0:8} {1:8} {2:8}".format(values[0], values[1], values[2]))
stu_grd.close()
```

Output

```
Name     Score    Grade
John     99       A
Ella     100      A
```

CSV files

- Write data into a CSV file

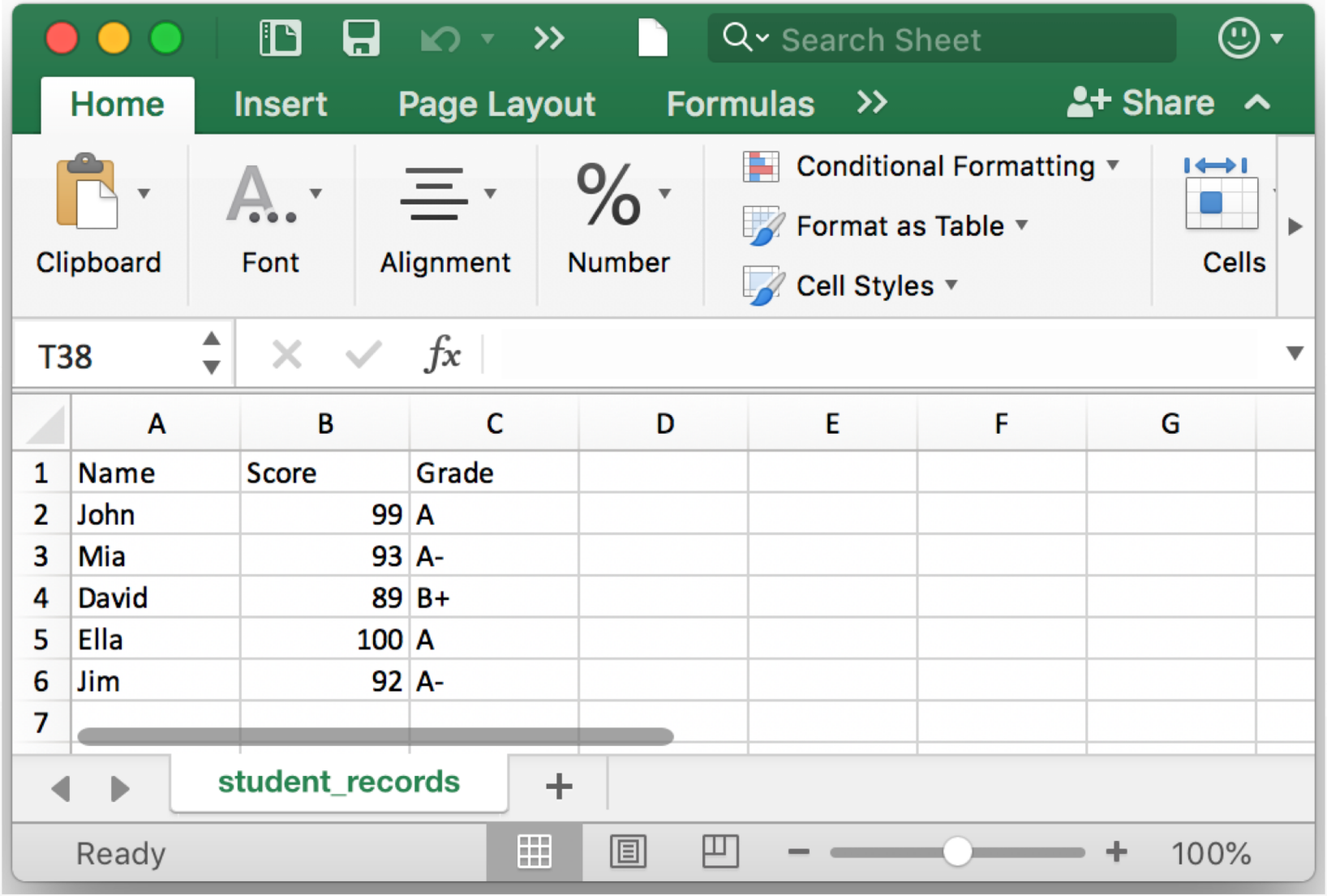
Codes

```
stu_rcds = [("John", 99, "A"),
            ("Mia", 93, "A-"),
            ("David", 89, "B+"),
            ("Ella", 100, "A"),
            ("Jim", 92, "A-")]

output_file = open("student_records.csv", "w")
# write the header
output_file.write('Name,Score,Grade')
output_file.write('\n')

# write each student's record
for stu_rcd in stu_rcds:
    rcd_string = '{},{},{}'.format(stu_rcd[0], stu_rcd[1], stu_rcd[2])
    output_file.write(rcd_string)
    output_file.write('\n')
output_file.close()
```

CSV files

- Write data into a CSV file

Output



Outline

- Reading and writing
- The *with* statement
- CSV files
- **Filenames and paths**

Filenames and paths

- Files are organized into directories, i.e., folders.
- For most operations in a running program, the default directory is the current directory.
- When the program opens a file, Python looks for it based on the given directory.
- The ***os*** module provides functions for working with files and directories.

```
>>> import os
>>> cwd = os.getcwd()
>>> cwd
'/home/dinsdale'
```

Return a string representing the current working directory

Filenames and paths

- A string like *'/home/dinsdale'* that identifies a file or directory is called a path.
- Relative path: A path that starts from the current directory.
- Absolute path: A path that starts from the topmost directory in the file system.
  - Can use *os.path.abspath* to find the absolute path to a file

```
>>> os.path.abspath('memo.txt')
'/home/dinsdale/memo.txt'
```

Filenames and paths

- Check whether a file or directory exists

```
>>> os.path.exists('memo.txt')
True
```

- Check whether it is a directory

```
>>> os.path.isdir('memo.txt')
False
>>> os.path.isdir('/home/dinsdale')
True
```

Filenames and paths

- The following function “walk” through a directory, prints the names of all the files, and calls itself recursively on all the directories.

```
def walk(dirname):
    for name in os.listdir(dirname):
        path = os.path.join(dirname, name)

        if os.path.isfile(path):
            print(path)
        else:
            walk(path)
```

os.listdir returns a list of the files (and other directories) in the given directory.

os.path.join takes a directory and a file name and joins them into a complete path.

Readings

- Think Python, chapters 9.1, 14.1, 14.2, and 14.4

References

- *Think Python: How to Think Like a Computer Scientist*, 2nd edition, 2015, by Allen Downey
- *Python Cookbook*, 3rd edition, by David Beazley and Brian K. Jones, 2013
- *Python for Data Analysis*, 2nd edition, by Wes McKinney, 2018
- *Fundamentals of Python Programming,* by Richard L. Halterman