INLS 560
Programming for Information Professionals

Files

UNC
SCHOOL OF INFORMATION
AND LIBRARY SCIENCE

Joan Boone
jpboone@email.unc.edu

Part 1: Overview

Part 2: Writing to files

Part 3: Reading from files

Why Use Files?

- Most applications use *persistent* data, i.e., data stored in files and databases
  - Read existing data from files/databases
  - Write (add, update, delete) data to files/databases for future use
- Types of files
  - Text files (contain readable text, such as Unicode)
  - Binary files (only readable by specific applications)
- Ways to access files
  - Direct (or random) access: similar to how video/audio players work – they can 'jump' directly to any data in the file
  - Sequential access: start at the beginning and read to the end of the file

File Names and File Objects

- Each operating system (Windows, Mac, Linux, etc.) has its own rules for naming files
  - In general, the convention is ***filename.extension***
  - Some typical file extensions (or file types)

    my_resume.pdf, **screenshot.png**,
    class_notes.txt, **hello_world.py**, **python.exe**

- File objects are used by programs to access files
  - A file object is created for a specific file and is stored in a variable that represents that object
  - The file object variable is used perform any operations on the file, e.g., to open, read, write, or close a file

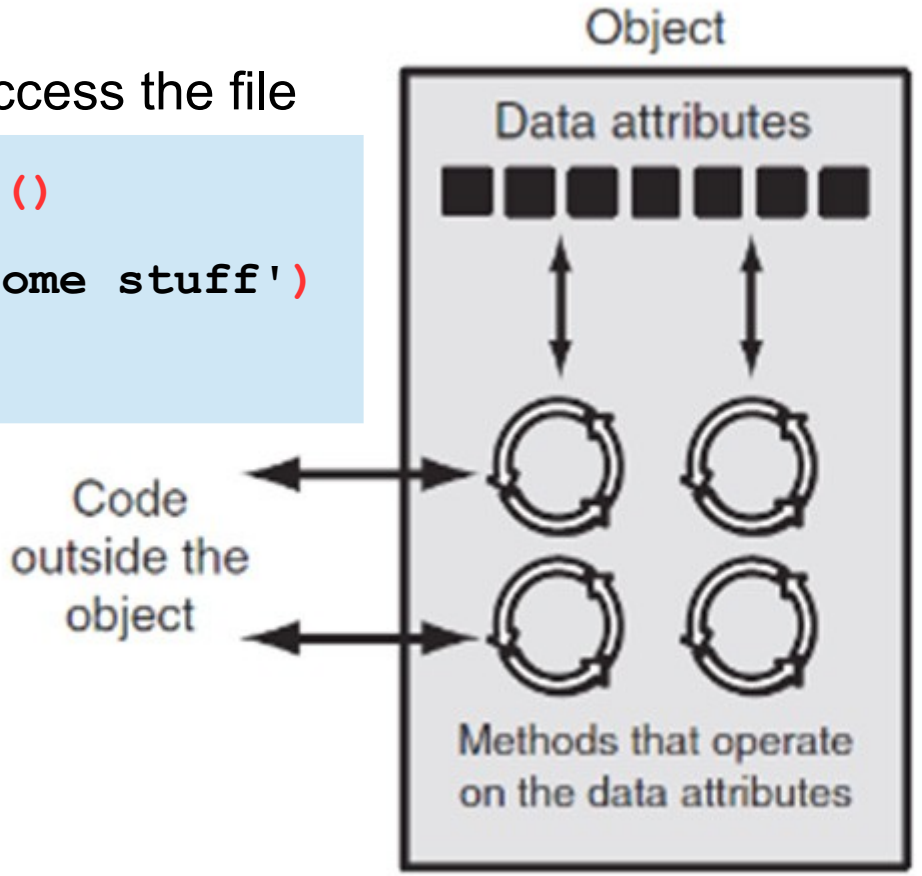
Object-oriented View of a File Object

When a you open a file, the `open` function returns a file object:

```
my_file = open(filename, mode)
```

Methods that access the file

```
my_file.readline()
my_file.write('some stuff')
my_file.close()
```



File attributes

```
name
mode
encoding
buffer
errors
...
```

Source: Starting Out with Python by Tony Gaddis

File Modes

- Before you can access a file, you have to open the file
- Use the `open` function to create a file object that is associated with a specific file

```
my_file = open(filename, mode)
```

  - `filename` is a string with the file name
  - `mode` specifies how the file is opened
  - `my_file` is the variable that references the file object
- Python file modes:

Source: Python documentation for 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 a disk file for updating (reading and writing)
`'U'`	*universal newlines* mode (deprecated)

File Modes

Frequently used file modes:

- `'r'` mode opens a file for reading only; the file cannot be changed or written to
- `'w'` mode opens a file for writing
  - Important: if the file already exists, its contents are erased and a new file is created.
  - If the file does not exist, it is created
- `'a'` mode opens a file for writing
  - Data written to the file is appended to the end of the file
  - If the file does not exist, it is created

If opening a file in the same directory as your program, you do not need to specify the path for the file name

```
test_file = open('test.txt', 'r')
```

Files in different directories

If opening a file in a different directory than where your program is located, you must specify the full directory path with the file name

Mac and Linux

```
test_file = open('/Users/Joan/temp/test.txt', 'w')
```

Windows

- If using forward slashes

```
test_file = open('C:/Users/Joan/temp/test.txt', 'w')
```

- If using backward slashes, 2 ways
  - use rawstring `r` prefix, so that Python interprets everything in the string as a literal character

```
test_file = open(r'C:\Users\Joan\temp\test.txt', 'w')
```

  - or, use double back slash to escape the back slash

```
test_file = open('C:\\Users\\Joan\\temp\\test.txt', 'w')
```

Part 1: Overview

Part 2: Writing to files

Part 3: Reading from files

Writing Data to a File

- Once the file is opened, string data can be written to it
- Use the **write** function with the file object to be written to

```
my_file.write(string_variable)
```

- Example

```
oscars = open('oscar_movies.txt', 'w')
oscars.write('Wings')     # literal string
movie_name = 'The Broadway Melody'
oscars.write(movie_name) # variable with string value
```

- After you have finished writing to the file, close the file:

```
oscars.close()
```

- Failure to close a file can cause a loss in data
  - Data is buffered before it is actually written. This improves performance because writing to memory is faster than writing to disk.
    When the buffer is full, it is written to the file (disk)
  - Closing the file ensures that any buffered data is written to the file

Writing Data to a File

```
# This program writes three lines of data to a file.
def main():
    # Open a file named oscar_movies.txt.
    oscars = open('oscar_movies.txt', 'w')

    # Write Oscar-winning movie names to the file
    oscars.write('Wings')
    oscars.write('The Broadway Melody')
    oscars.write('All Quiet on the Western Front')

    # Close the file.
    oscars.close()

# Call the main function.
main()
```

Contents of file, **oscar_movies.txt**:

```
WingsThe Broadway MelodyAll Quiet on the Western Front
```

Writing Data to a File

```
# This program writes three lines of data to a file.
def main():
    # Open a file named oscar_movies.txt.
    oscars = open('oscar_movies.txt', 'w')

    # Write Oscar-winning movie names to the file
    oscars.write('Wings\n')
    oscars.write('The Broadway Melody\n')
    oscars.write('All Quiet on the Western Front\n')

    # Close the file.
    oscars.close()

# Call the main function.
main()
```

Use the newline escape sequence, **\n** , to separate items in the file:

File contents:

```
Wings
The Broadway Melody
All Quiet on the Western Front
```

file_write.py

Exercise: Create a simple web page

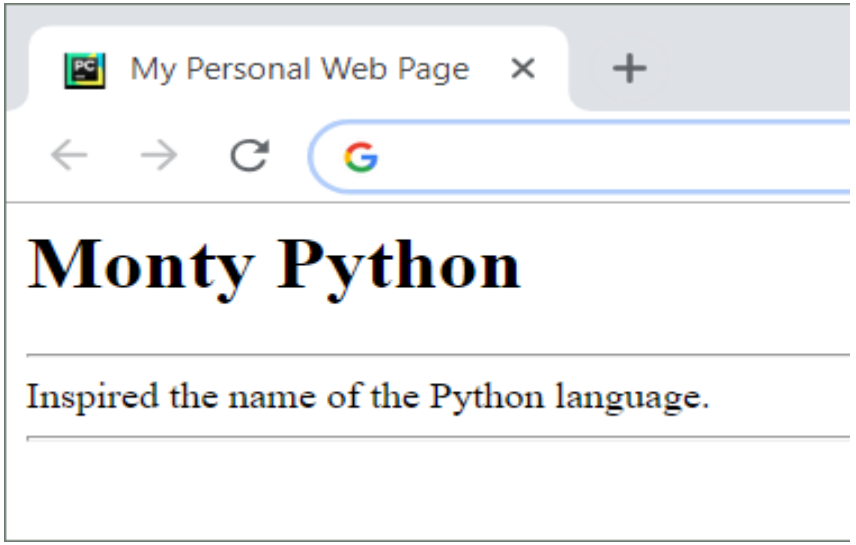
- Prompt user for name and description
- Create an HTML file
- Write the HTML with the name and description values

```
Enter your name: Monty Python
Describe yourself: Inspired the name of the Python language.
```

my_page.html

```
<html>
<head>
    <title>My Personal Web Page</title>
</head>
<body>
    <h1>Monty Python</h1>
    <hr>
    Inspired the name of the Python language.
    <hr>
</body>
</html>
```

Displayed in browser



webpage_generator_draft.py

Part 1: Overview

Part 2: Writing to files

Part 3: Reading from files

Reading Data from a File

The **read** method returns the entire contents of the file as a string

```
# This program reads the contents of a file
def main():
    # Open a file named oscar_movies.txt.
    oscars = open('oscar_movies.txt', 'r')

    # Read the file's contents
    file_contents = oscars.read()

    oscars.close()          # Close the file.

    print(file_contents)    # Print the contents

# Call the main function.
main()
```

Output:
```
Wings
The Broadway Melody
All Quiet on the Western Front
```

file_read.py

Reading Data from a File, one line at a time

The **readline** method reads one line at a time from the file as a string

```
# This program uses readline to read the file contents
def main():
    # Open a file named oscar_movies.txt.
    oscars = open('oscar_movies.txt', 'r')

    # Read three lines from the file
    line1 = oscars.readline()
    line2 = oscars.readline()
    line3 = oscars.readline()

    oscars.close()   # Close the file.

    print(line1)
    print(line2)
    print(line3)

main()
```

Output:

```
Wings

The Broadway Melody

All Quiet on the Western Front
```

file_readline.py

Removing the Newline Character

Use **rstrip** method to remove **\n** from the (right) end of the string

```
# This program uses rstrip to remove newline
def main():
    oscars = open('oscar_movies.txt', 'r')

    line1 = oscars.readline()
    line2 = oscars.readline()
    line3 = oscars.readline()

    # Strip the \n from each string.
    line1 = line1.rstrip('\n')
    line2 = line2.rstrip('\n')
    line3 = line3.rstrip('\n')

    oscars.close()

    print(line1)
    print(line2)
    print(line3)

main()
```

Returns a copy of the string with trailing newline character removed

Output:

```
Wings
The Broadway Melody
All Quiet on the Western Front
```

rstrip_newline.py

Writing Numeric Data to a File

- Strings can be written directly to a text file, but numbers must be converted to strings before they can be written to a file.
- Built-in function, **str**, converts a value to a string.

```
def main():
    outfile = open('numbers.txt', 'w')

    # Get three numbers and calculate the sum
    num1 = float(input('Enter a number: '))
    num2 = float(input('Enter another number: '))
    num3 = float(input('And one more number: '))
    sum = num1 + num2 + num3

    # Write the numbers to the file.
    outfile.write(str(num1) + '\n')
    outfile.write(str(num2) + '\n')
    outfile.write(str(num3) + '\n')

    outfile.close()  # Close the file
    print('Sum =', format(sum,'.2f'))  # Print the sum

main()
```

write_numbers.py

Reading Numeric Data from a File

- When reading numbers from a file, you must convert them from strings to numbers before you can use them in arithmetic expressions
- Built-in functions, **int** and **float**, convert strings to numbers

```
# This program reads numbers from a file,
# converts them to numbers and calculates product

def main():
    infile = open('numbers.txt', 'r')

    # Read three numbers from the file.
    num1 = int(infile.readline())
    num2 = int(infile.readline())
    num3 = int(infile.readline())

    infile.close()

    # Multiply the three numbers.
    product = num1 * num2 * num3

    # Display the numbers and their total.
    print('The numbers are:', num1, num2, num3)
    print('Their product is:', product)

main()
```

read_numbers.py

Using Loops to Write to a File

Files generally contain a lot of data, so programs often use loops to read and write the contents.

```
def main():
    total_sales = 0

    # Get the number of days.
    num_days = int(input('For how many days do you have sales? '))

    # Open the file for writing
    sales_file = open('sales.txt', 'w')

    # Get the sales amount and write to the file
    for count in range(1, num_days + 1):
        # Get the sales for a day, and add to running total
        sales = float(input('Enter the sales for day #' + str(count) + ': '))
        total_sales = total_sales + sales

        # Write the sales amount to the file.
        sales_file.write(str(sales) + '\n')

    sales_file.close()   # Close the file
    print('Total sales is', format(total_sales, '.2f'))  # Print total sales

main()
```

Source: Starting Out with Python by Tony Gaddis

write_sales.py

Using `for` Loop to Read a File (recommended approach)

- Python also provides a simple approach for looping through a file by using a `for` loop that automatically reads a line in a file without checking any special end-of-file (EOF) conditions
- The loop iterates once for each line in the file

```
# This program reads all of the lines in the sales.txt file.
def main():
    sales_file = open('sales.txt', 'r')

    # Read all the lines from the file.
    for line in sales_file:
        # Convert line to a float.
        amount = float(line)
        # Format and display the amount.
        print(format(amount, '.2f'))

    sales_file.close()

main()
```

Exercise: Reading Data from a File

- Update the `average_steps.py` program to read the numbers from the file, `steps.txt`. Each number represents the number of steps taken on each day of the year.
- Calculate the total number of steps
- Keep a counter of the number of lines (days) read
- Display the average number of steps taken per day. Output should be:

```
The average number of steps taken in 365 days
was 5,296.8
```

average_steps_draft.py