Review

www.pollev.com/python002

Working with the Web and an Introduction to File Input & Output

Introduction to Programming - Python

Exceptions

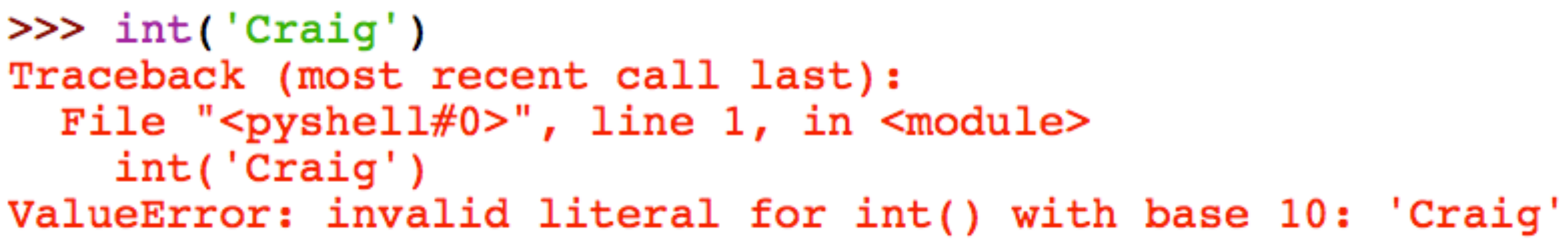
Exceptions

- An exception is any error that causes a program to halt while it's running.
- Example:

```
x = 'Craig'
y = int(x)
```

+
Exceptions

- When an exception occurs Python will generate a "traceback"
- Tracebacks give information about the exception that occurred, including the line number that caused the issue and the name of the exception
- Programmers generally say that an exception has been "raised" by the program



```
>>> int('Craig')
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    int('Craig')
ValueError: invalid literal for int() with base 10: 'Craig'
```

Preventing Exceptions

- Many times you can avoid exceptions all together by adjusting how we code our algorithms. For example, the following code will raise an exception if x = 0

```
x = int(input('give me a number'))
print (100/x)
```

- Yet we can avoid the issue by wrapping the potentially dangerous code inside a selection statement:

```
x = int(input('give me a number'))

if x != 0:
   print (100/x)
```

+
Catching Exceptions

- Python has an exception handling statement that can be used to “catch” exceptions and prevent them from crashing your program

- Syntax:

```
try:
   # questionable code goes here
except:
   # this block will run if the code in the
   # ‘try’ block above raised an exception
else:
   # this block will run if the code in the
   # ‘try’ block above was successful
```

Data Validation using Try / Except

- You can use the Try / Except statement to validate the data type of inputted data
- Programming challenge:
  - Ask the user for a test score as well as the total # of points available on the test
  - Ensure that the test score provided is a valid floating point number
  - Ensure that total points is a valid floating point number
  - Generate the test average. Make sure you don't divide by zero and raise an exception.

+

What will be the result of the following program?

```
try:
    num1 = 6
    num2 = 0
    answer = num1 / num2
except:
    print("Division by Zero!")
else:
    print("The answer is:", answer)
```

What will be the result of the following program?

```
try:
   number = int(input("Enter a number: ")) # user enters 4.0
   answer = number * number
except:
   print('Invalid input.')
else:
    print('The answer is', answer)
```

Strings and Lists

+

Splitting a string

- Sometimes you are given long strings of information that need to be "parsed" in a particular way
- For example, consider the following string:

```
x = "Craig,Professor,NYU"
```

- This string contains three pieces of data – a name, a title and an employer. The data has been "packaged" into a single string in a meaningful way (i.e. we know that the comma character separates the different data points)

+
Splitting a string

- We can use a technique called "splitting" to "unpack" a string and extract its individual values. The trick is to isolate a "separator" character that can be used to delimit the data you want to extract. Here's an example:

```
mystring = "Craig,Professor,NYU"

splitstring = mystring.split(",")

print (splitstring)

>> ['Craig', 'Professor', 'NYU']
```

Splitting a String

- General Algorithm:
  - Obtain a string that contains data organized in a certain way
  - Split the string into a list
  - Process the list using list mechanics

+

Programming Challenge

- The following string represents the test scores of a given student.

```
scores = “95,100,67,33,88”
```

- Write a program that:
  - Total # of scores
  - Prints out the average score for the student
  - Prints out the highest and lowest scores
  - Drops the lowest score and prints out the student’s average without that score

+

Working with External Data

Obtaining Data from the Web

- You generally won't be "hard coding" data into your programs. You will usually be processing information that comes to you in the form of user input or via an external data source.

- One way to easily get information into Python from the "outside world" is to initiate a connection to the Web and obtain data via a URL

+
Obtaining Data from the Web

- General Algorithm:
  - Initiate a web connection to an external data source
  - Obtain the data that exists at that location as a String
  - “Clean up” the string into a meaningful form that your program can understand. This usually involves making sure that the character set used in the external file is one that your computer can handle.
  - Parse your string into a list
  - Process this list and generate output

+
Working with the web

- You can initiate web connections using the "urllib" module. Example:

```
import urllib.request

# where should we obtain data from?
url = "https://www.cnn.com/ "

# initiate request to URL
response = urllib.request.urlopen(url)

# read data from URL as a string
data = response.read().decode('utf-8')

print (data)
```

Warnings!

- The web is a volatile place – servers can go up and down at a moment's notice, and your connection to the network may not always be available
- You need to be prepared to deal with connection errors in your programs.
- You can use the try / except / else suite to help "catch" errors before they crash your program.

If you are experiencing issues with urllib.request. ...

- In IDLE you will need to explicitly install a "security certificate" to allow your program to make a connection to a website. You can set this up by doing the following (note that you only need to do this once):

- Click on Finder on a Mac, or File Explorer if on Windows

- Navigate to the directory where Python is installed. On a Mac it will be under the Applications folder. On a PC it will most likely be under C:\Program Files

- Look for the the "Install certificate.command" file -- double click on this file to run it

- You should now be able to make web connections via the urllib.request library in your programs

+
Programming Challenge

- Write a Python program that asks the user for a name.
- Determine if that name is one of the most 50 popular names for boys or girls in the United States in 2020.
- There are two data files on the web which represent the most popular boy and girl baby names from 2020 according to the US Social Security website. These names are sorted in popularity order (i.e. the first item is the most popular name in each file).

  https://text-files.glitch.me/girlNames.txt

  https://text-files.glitch.me/boyNames.txt

Programming Challenge

- This data file contains all World Series winning teams up until a few years ago:

  https://text-files.glitch.me/worldSeries.txt

- Write a program that reads in this data and finds the team that won the most games

+

File Input & Output (I/O)

Programs and Memory

- All of the programs that you have been writing so far have reset themselves each time they run
- This is because the data associated with the program (i.e. your variables) are stored in your computer's memory (RAM)
- RAM is a volatile place – it serves as a computer's short term memory. RAM that is being used by a program is cleared when the program stops running.
- If programs are to retain data between executions then we must find a more permanent way to store and save information for future use.

Long term storage

- A computer almost always has at least one type of long-term storage device at its disposal (usually some kind of hard drive)
- We can use the long term storage capabilities of a computer to store data in the form of a file.
- Once we save a file it will remain on the long term storage device after the program is finished running, and can be accessed and retrieved later on.

Long term storage

- This is a pretty common technique and is used by almost all programs that need to keep track of some kind of information between executions.
- Examples:
  - Application software (Word, Photoshop, etc)
  - Gaming
  - The Web (cookies)

+

File Types

- There are two main families of files that we work with when writing programs
  - Text Files
  - Binary Files

Text Files

- Text files are files that contain data that is encoded as text (i.e. ASCII or Unicode characters)
- Data stored in a text file is visible and can be read by a program that is designed to view / edit textual data (i.e. a word processing program)
- Examples:
  - Notepad documents
  - Cookies
  - HTML files
  - XML files
- We will be working with text files exclusively in this class

Binary Files

- Binary files contain data that is not encoded as a text format
- Binary files are intended to be read by other programs and not by humans directly
- Binary files appear as "gibberish" when viewed via a word processing program
- Examples
  - Image files (PNG / GIF / JPG files)
  - Movie Files (MOV / MP4 / AVI files)
  - Application files (EXE / APP files)

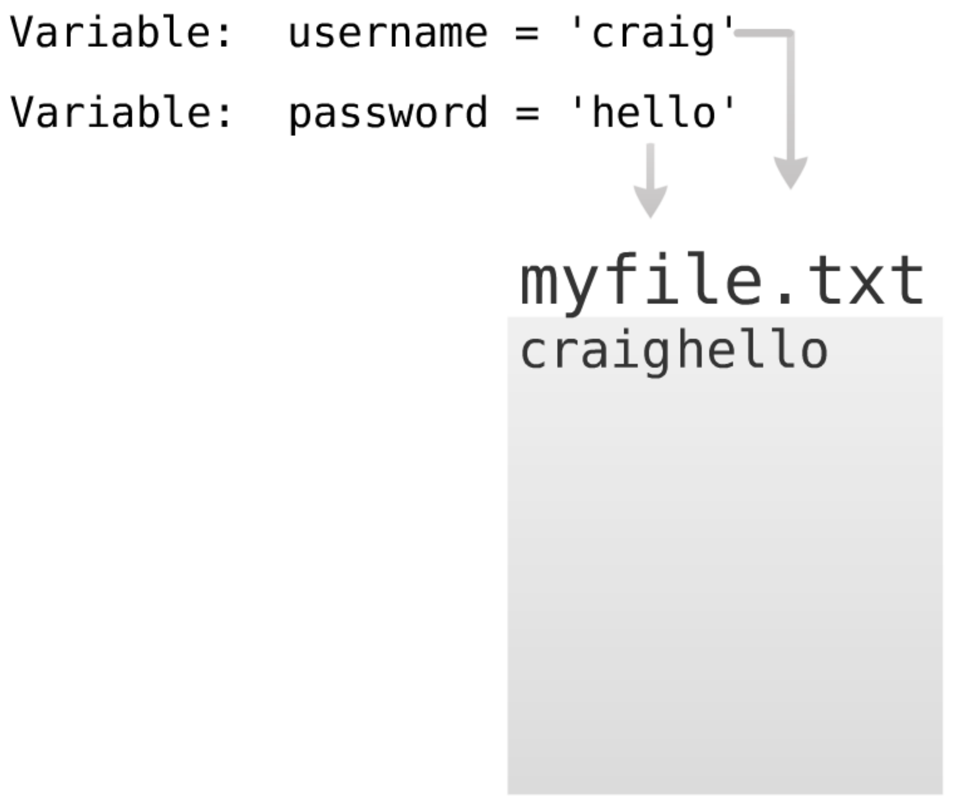
Working with files

- Step 1: Open up a connection between your program and the operating system. Define the name and location of the file you wish to work with as well as what you want to do to that file (read or write)

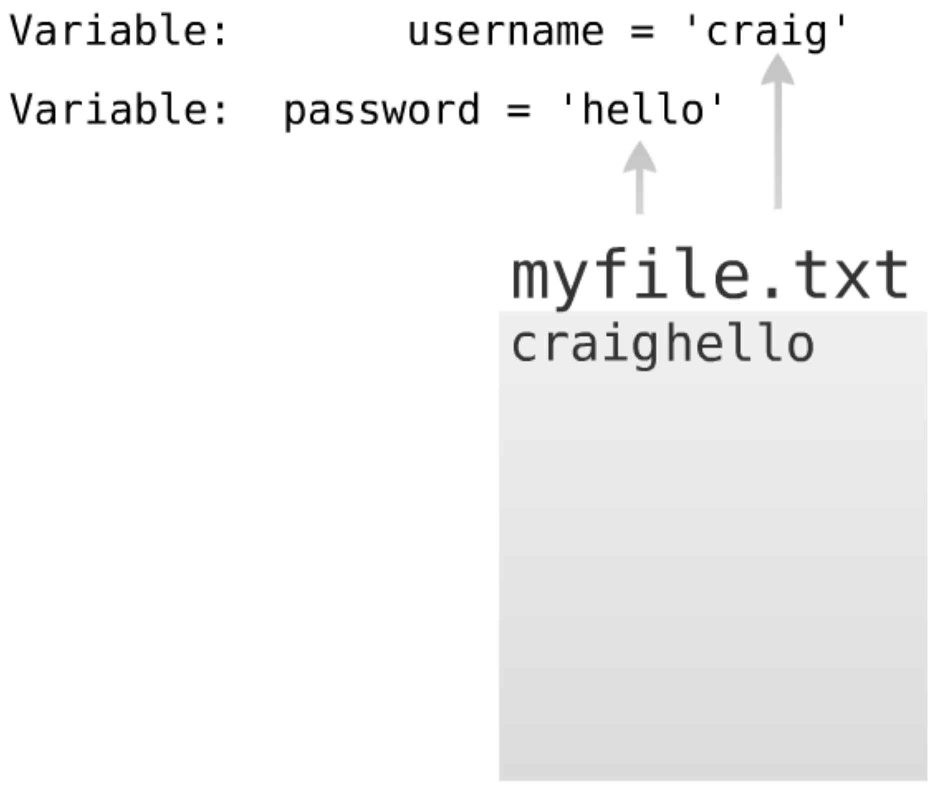
- Step 2: Process the data
  - Writing: send information in the form of variables from your program into the file to be written
  - Reading: extract information into variables in your program

- Step 3: Close the file, telling the OS to release the file for use by other programs

Writing information to a file



Reading information from a file



File Access Methods

- Sequential Access: You must read a file starting at the beginning and continue until you reach the data you are looking for (similar to how a cassette tape works) – this is the method we will be focusing on in this class

- Random Access: You can skip from point to point in a file, allowing you to jump over data until you find what you are looking for (similar to how a CD works)

+
Filenames and File Objects

- Filenames are nothing more than labels that allow us to uniquely identify groupings of data that exist on a long term storage device

- Every file must have a filename

- On most operating systems a filename comes with a "file extension" which allows you to quickly tell what kind of data is stored inside that file. Extensions come after the last period in a file (i.e. "mydocument.doc" – "doc" is the file extension)

- File extensions tell the operating system what kind of data is stored in the file and allows the OS to select an appropriate program to open a particular file

Filenames and File Objects

- A File Object is a special variable that you must create in a programming language that serves as a connection between your program and the operating system
- File Objects tell the OS the name of a file that you wish to work with along with what you want to do to that file (write, read, append, etc)

+
Opening a file in Python

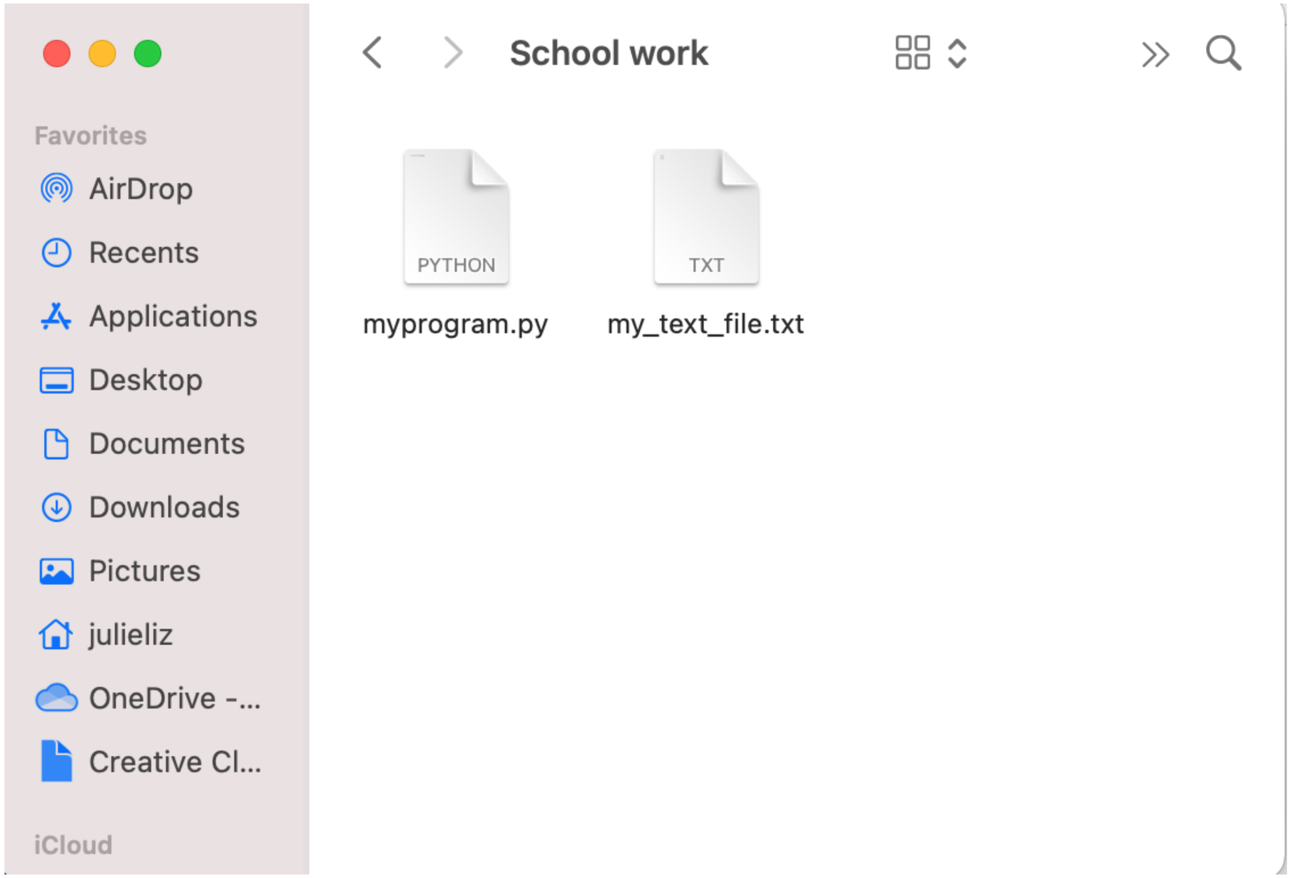
```
file_object = open(filename, mode)

# File Modes
# 'r' = read
# 'w' = write - if the file does not exist, create it
#             - if the file does exist, overwrite it
# 'a' = append - if the file does not exist, create it
#              - if the file does exist, append data to the end of it
```

+
Opening a file in Python

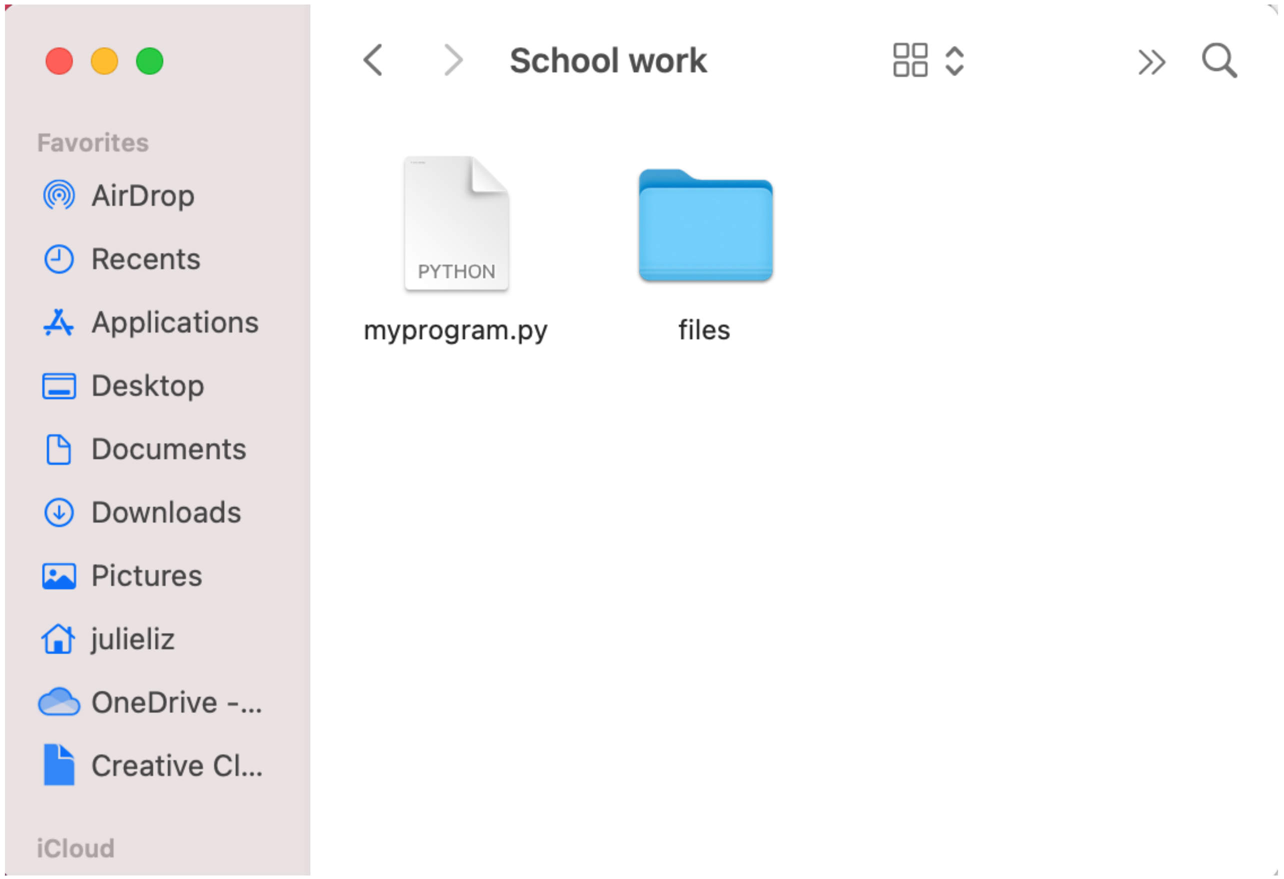
- Filenames are expressed as strings
- If you type in a simple filename (i.e. “myfile.txt”) you are telling Python to open up a file with that name in the current directory
- You can ask Python to open up a file that exists outside of the current directory by typing in an absolute path. For example:
  - Windows: ‘C:\temp\testfile.txt’
  - Mac: ‘/var/local/testfile.txt’
- When providing an absolute path you should preface the string with the ‘r’ directive – this tells Python to treat the text in the string as “raw” text and not try and add in any escape characters. Example:
  - `file_object = open(r’C:\temp\testfile.txt’, ‘w’)`

+
File Path Practice



The path to the txt file will be: "my_text_file.txt"

+
File Path Practice



The path to the txt file will be: “files/my_text_file.txt”

+

Writing data to a file

- You can write data into a file once you have created a file object and have opened a file for writing using the write() function. Here's an example:

```
file_object = open(‘myfile.txt’, ‘w’)

file_object.write(‘craig’)
file_object.write(‘hello’)

# close the file when you’re done
file_object.close()
```

+

Programming Challenge

- Prompt the user for a username and a password
- Store the username and password into a file named "security.txt"

+
Delimiters & Files

- You want to try and avoid writing files that concatenate all of your data into one long line of unintelligible text
- This is bad practice since it can be very difficult – or impossible – to extract out your data later on
- One way to separate data in a text file is by splitting out your data into multiple lines using the "\n" escape character. This allows you to store a single value on each line of your text file, which will make things a lot easier for you later on when you need to read your file and perform some kind of operation on the data contained within

+
Programming Challenge

- Prompt the user for a username and a password
- Store the username and password into a file named "security.txt"
- Store each value on its own line

+ Reading data from a file

- You can read data contained inside a file once you have created a file object and have opened a file for reading using the read() function. Here's an example:

```
myvar = open("test.txt", "r”)

alldata = myvar.read()
print (alldata)

myvar.close()
```

The read() function

- The read() function extracts all data from a file as a string and returns a string
- It returns one large string that must be further processed before it can actually be used

+

Programming Challenge

- Write a program that opens your "security.txt" file and reads in the username and password store in the file
- Store these values into a series of variables
- Prompt the user for a username and password
- If they match the values stored in the file, allow them to continue. Otherwise present an error message.

Writing Numeric Data to a file

- The write() function only accepts strings – the following code will generate a runtime error:

```
myvar = open("test2.txt", "w”)
myvar.write(55)
myvar.close()
```

- You need to convert any data that will be stored in a file into a string by using the str() conversion function. Example:

```
myvar = open("test2.txt", "w”)
myvar.write(str(55))
myvar.close()
```

+

Reading Numeric Data from a file

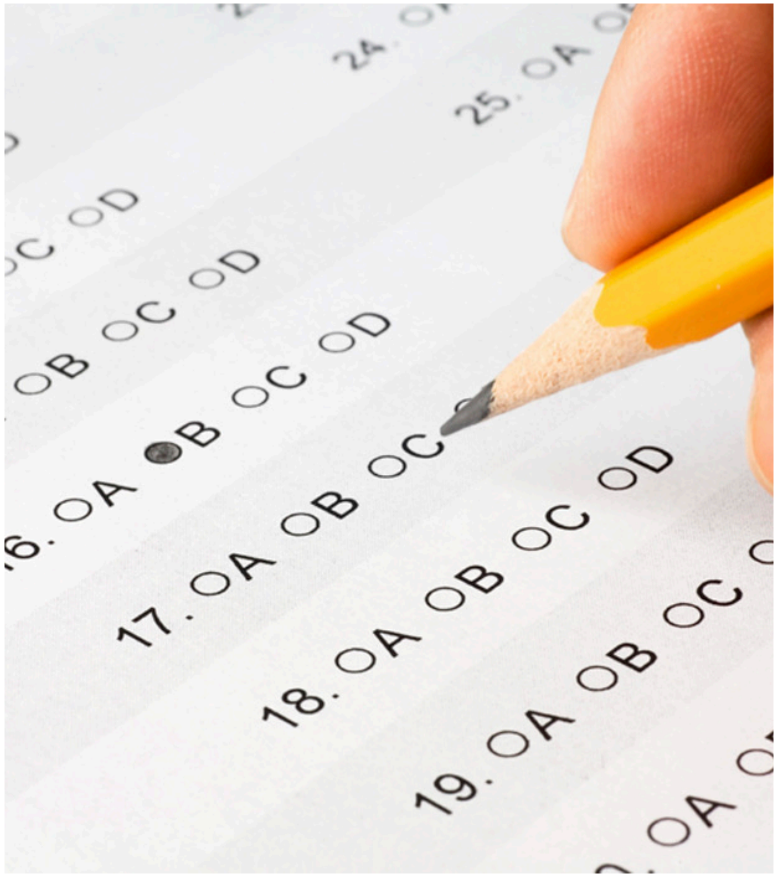
- When you are reading data from a file you will need to convert strings to ints or floats if you want to perform calculations on them
- This is identical to what we have to do when using the input() function to ask the user to enter a string
- Note: The int() and float() functions automatically remove line break characters for you!

+
Programming Challenge

- Write a program that opens up a file named "testscores.txt". This file contains the following information in the following format:

```
student name
score1
score2
score3
```

- Read in the values and print out the average score for the student specified in the file along with the student's name.



Using loops to write data to a file

- You can use the write() function inside a repetition structure to write multiple values to a file

Programming Challenge

- Continually prompt a user for a series of price values
- Store these values in a text file called "prices.txt"
- When the user enters a price value of 0 or less you can assume they want to end the program
- If the user runs the program more than once you should not overwrite the previous text file – simply append the new price values to the end of the file
- Hint: open the file using the 'a' flag

Programming Challenge

- Open your ‘prices.txt’ file for reading
- Read in all values and calculate the total price based on the values that are contained in the file

Programming Challenge

- Write a "matchmaking" program that asks the user to enter in their name, favorite color and favorite food.
- Store the result in a text file

eharmony

Programming Challenge

- Interface with your matchmaking text file and ask a second user for a favorite color and favorite food.
- Compare the results – if they get 0/2 questions correct, they are not a match! ½ , they might be a match. 2/2, they are definitely a match!

+

Files & Lists

Storing lists in Files

- You can store list data in a file by doing the following:
  - Open up a file
  - Iterate through your list
  - Write each value to the file, making sure to convert each value to a string and append a line break character to each item

Programming Challenge

- Write a program that asks the user to enter in a series of questions in the following format:

  ```
  Enter a question:  What color is the sky?
  Enter an answer:  Blue
  Would you like to enter another question? (y/n):  n
  ```

- Write the questions and answers to a text file

+

Programming Challenge

- Write a program that interfaces with your question text file and prompts the user for the answer to each question
- If they get a question right you can give them a point. Otherwise they do not earn a point.
- At the end of the program you can display the total number of points earned as well as their score (i.e. 5/10 questions correct = 50%)
- Extension: randomize the quiz for the user

Programming Challenge

- Open the "excel_gradedata.xlsx" file and notice its format
- Save it as a text file (CSV)
- Write a Python program that opens it up and prints out the average score for each student that is represented in the file

Programming Challenge

- Write a program that asks the user for a filename
- Then ask the user for a word to replace and its replacement
- Replace all occurrences of the word and write the result in a new file using the "_replaced" suffix (i.e. if the user enters the filename "test.txt" you would create a file called "test_replaced.txt"