Intro to Computer Science CS-UH 1001, Spring 2022

Lecture 11 – File I/O, CSV Files, Exception Handling

Today’s Lecture

- File Input/Output
- Comma Separated Values Files (CSV)
- Exception Handling

Recap

- Dictionaries:
  - are defined with {}
  - every element in a dictionary is a key-value pair
    - grades = {‘Jon’:90, ‘Robb’:80, ‘Arya’:60}

- keys are unique
- keys must be an immutable data type

- Unlike Lists, dictionaries elements are not stored in order

Files

- So far, we have only variables to store data
- Variables are stored in memory
  - Once the program is terminated, variables are cleared and their data is lost
- If we want to keep data even if the program terminates, the data has to be stored in a file

We use files every day

- We use files on a day-to-day basis to store different data in our computers
  - Image or videos
  - MS Word documents
  - Emails
  - Games
  - Homework assignments

- Most of the programs that we use store data in files

File Types

- Generally, two types of files:
  - binary: images, audio, videos, MS Word, executables, etc.
  - text: files that contain characters
- Text files store data as text using a certain coding scheme like ASCII
  - We can easily open it with any text editor
- Binary files store data as 0 and 1
  - The data stored are intended for the program and we can not open it in a text editor

File Types

- We will only focus on text files

- We will be able to open these files in any text editor and look at their content

Opening a File

- Syntax:

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

- open() function returns a file object
- filename is a string specifying the file to open
- mode specifies how to open the file
  - 'w' to open a file for writing
  - 'a' to open a file for writing and appending to it
  - 'r' to open a file for reading only (you can not write to the file)

‘w’ vs. ‘a’ Modes

- Both ‘w’ and ‘a’ modes open a file for writing

- ‘w’:
  - If the file exists, it will erase/overwrite its content
  - If the file does not exist, it will create it

- ‘a’:
  - If the file exists, it will keep its content and appends new data to the end of the file
  - If the file does not exist, it will create it

‘r’ Mode

- The ‘r’ mode opens the file in read-only mode
  - if the file does not exist, an error is thrown

```
input_file = open("my_file.txt", ‘a’)
input_file.close()
input_file = open("my_file.txt", ‘r’)

# read data from the file

input_file.close()
```

Workaround: Creates a file if it did not exist and closes it

Don’t forget to close the file!

- When a program opens a file, you always have to close it at the end once you finished writing
- If you don’t close the file, you run the risk that your writings might not be recorded in the file
- To close the file, use the .close() method as:

```
file_variable.close()
```

Writing to a File

- After opening a file in write mode (‘w’ or ‘a’), use the .write() method to write to the file:

- Example:

```
output_file = open(‘my_file.txt’, ’w’)
output_file.write(“a”)
output_file.write(“b”)
output_file.write(“c”)

output_file.close()
```

my_file.txt

```
abc
```

Example: Writing to a File

```
output_file = open(‘my_file.txt’, ‘w’)

output_file.write(‘a\n’)
output_file.write(‘b\n’)
output_file.write(‘c\n’)

output_file.close()
```

my_file.txt

```
1 a
2 b
3 c
4
```

Reading from a File

- Everything that is read from a file is a string!
- There are two methods for reading content from a file:
  - .read():
    - reads the entire file at once
    - Use this with caution, because if the file is bigger than your memory space, you might run into problems
  - .readline():
    - Reads a single line from the file, including the \n
    - Every subsequent call will read the next line
    - Once you reach the last line it will return a ‘’

Example: Reading from a File

```
input_file = open(‘my_file.txt’, ‘r’)

file_content = input_file.read()
# file_content: ’a\nb\nc\n’

print(file_content)# ’a\nb\nc\n\n’
print(‘Good bye’)
input_file.close()
```

my_file.txt

```
1 a
2 b
3 c
4
```

Terminal output:

```
a
b
c

Good bye
```

The print() adds a \n

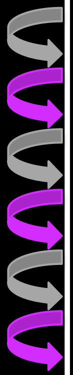
Example 2: Reading from a File

```
input_file = open(‘my_file.txt’, ‘r’)

line_1 = input_file.readline()
# line_1: ‘a\n’
line_2 = input_file.readline()
# line_2: ‘b\n’
line_3 = input_file.readline()
# line_3: ‘c\n’

input_file.close()

print(line_1) # ‘a\n\n’
print(line_2) # ‘b\n\n’
print(line_3) # ‘c\n\n’
print(‘Good bye’)
```

my_file.txt

```
1 a
2 b
3 c
4
```

Terminal output:



```
a

b

c

Good bye
```

String strip() Method

- The .strip() method returns a copy of the string with both leading and trailing characters removed, i.e. “\n”, “ “, etc

- Example:

```
string = “     Hello World!                    \n”
string = string.strip()
print(string)               “Hello World!”
```

Example 3: Removing the \n?

```
input_file = open(‘my_file.txt’, ‘r’)

line_1 = input_file.readline().strip() # ‘a’
line_2 = input_file.readline().strip() # ‘b’
line_3 = input_file.readline().strip() # ‘c’

input_file.close()

print(line_1) # ‘a\n’
print(line_2) # ‘b\n’
print(line_3) # ‘c\n’
print(‘Good bye’)
```

my_file.txt

```
1 a
2 b
3 c
4
```

Terminal output:

```
a
b
c
Good bye
```

Looping through Files

- Remember that for-loops can loop through sequences!?
- A file object is also a sequence!

```
input_file = open('my_file.txt', 'r')
for line in input_file:
    print(line.strip())

input_file.close()
print('Good bye')
```

my_file.txt

```
1 a
2 b
3 c
4
```

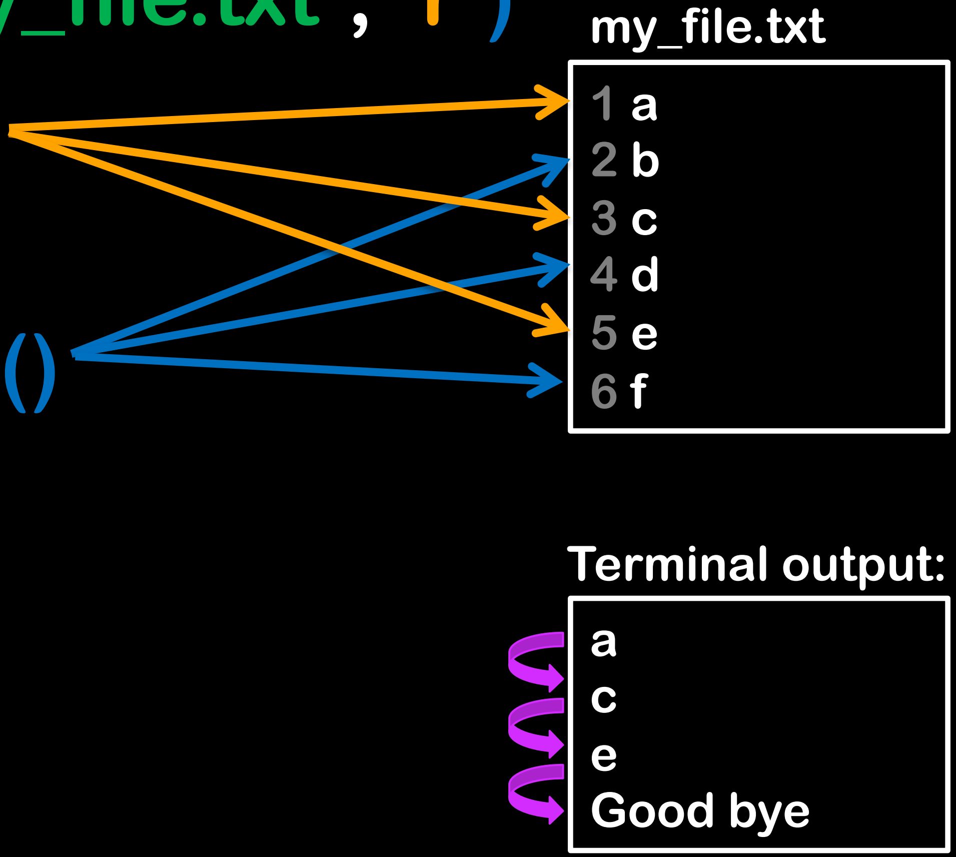
Terminal output:

```
a
b
c
Good bye
```

Is the .readline() missing?

Looping through Files

```
input_file = open('my_file.txt', 'r')
for line in input_file:
    print(line.strip())
    input_file.readline()

input_file.close()
print('Good bye')
```



my_file.txt

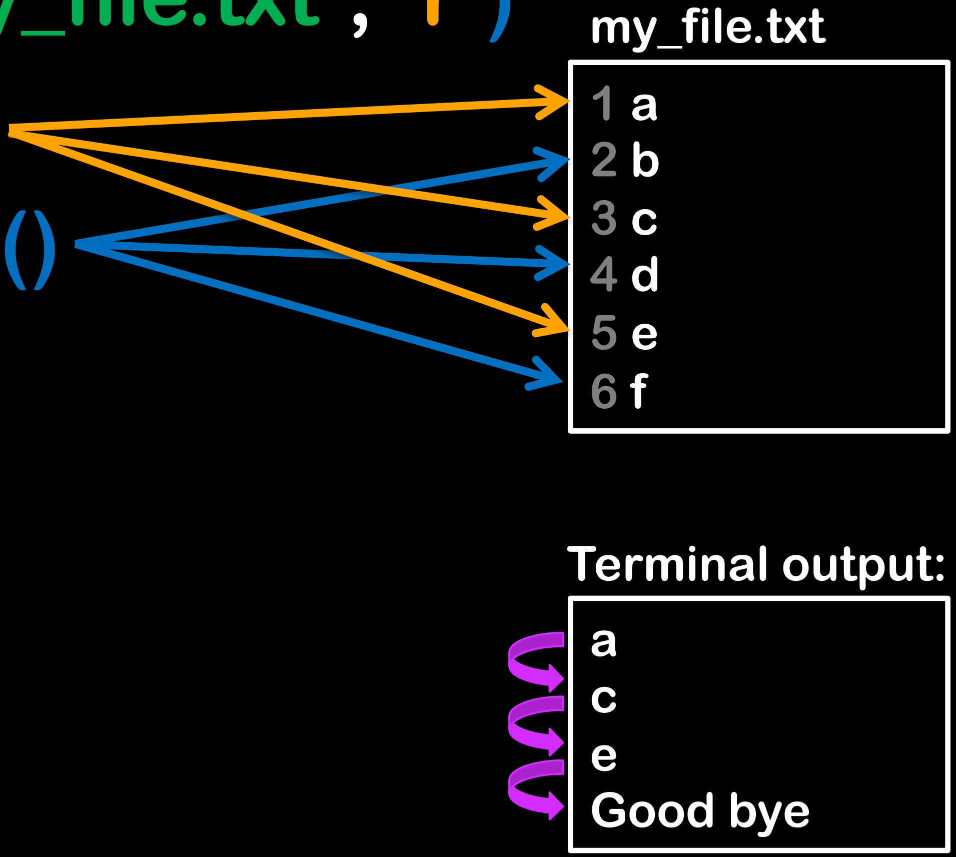
```
1 a
2 b
3 c
4 d
5 e
6 f
```

Terminal output:

```
a
c
e
Good bye
```

Looping through Files

```
input_file = open(‘my_file.txt’, ‘r’)
for line in input_file:
    input_file.readline()
    print(line.strip())

input_file.close()
print(‘Good bye’)
```



my_file.txt

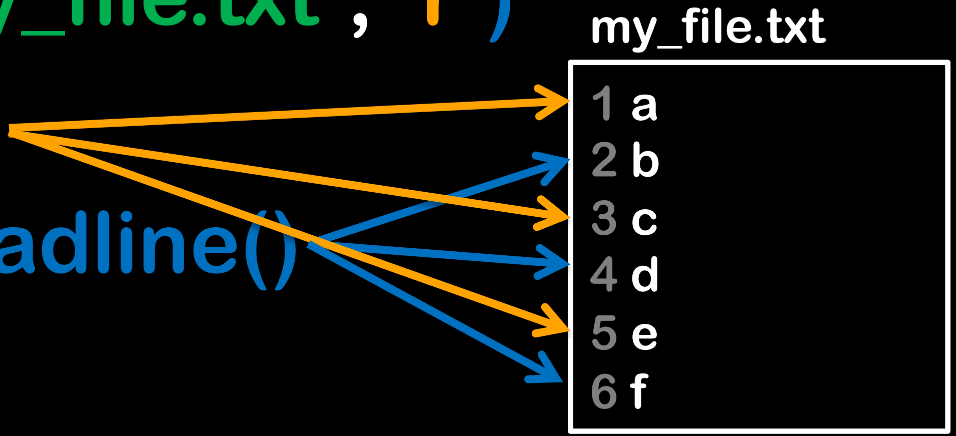
```
1 a
2 b
3 c
4 d
5 e
6 f
```

Terminal output:

```
a
c
e
Good bye
```

Looping through Files

```python
input_file = open('my_file.txt', 'r')
for line in input_file:
    line = input_file.readline()
    print(line.strip())

input_file.close()
print('Good bye')
```



my_file.txt

```
1 a
2 b
3 c
4 d
5 e
6 f
```

Terminal output:

```
b
d
f
Good bye
```

CSV Files

CSV Files

- Comma Separated Value (CSV) files are simple text files that contain records of data
  - The records are separated by line breaks (/n)
  - Each record consists of fields separated by some character (usually a comma)
- Each record usually has the same sequence of fields

CSV files

Tabular data

$Value_{1,1}$	$Value_{1,2}$		$Value_{1,n}$
$Value_{2,1}$	$Value_{2,2}$		$Value_{2,n}$
. . .			
$Value_{m,1}$	$Value_{m,2}$		$Value_{m,n}$

CSV file

$Value_{1,1}$, $Value_{1,2}$, …, $Value_{1,n}$
$Value_{2,1}$, $Value_{2,2}$, …, $Value_{2,n}$
.
.
.
$Value_{m,1}$, $Value_{m,2}$, …, $Value_{m,n}$

Read and write CSV files

- Remember the string methods:
  - .join()
  - .split()

- These two methods come in very handy when working with CSV files and Lists

  - Writing: use the string method ‘,’.join(list) to create one comma-separated line to store in a CSV file

  - Reading: use the string method .split(‘,’) to split one CSV line directly into a list with elements

Hands-on Session
Birthday App 2.0

Birthday app 2.0 (ex_11.1.py)

- Upgrade the Birthday program to
  1. save the dictionary data to a CSV file when you quit the program
     - write dictionary elements to the file, e.g. key,value
  2. read the data from a CSV file when you start the program
     - populate the dictionary from the file
     - Hint: use the following lines to create the file if it does not exist, otherwise open(”birthdays.csv", ”r") will throw an error

```
in_file = open(”birthdays.csv", "a")
in_file.close()
in_file = open(”birthdays.csv", ”r")
```

Exception Handling

Exceptions

- Exception are runtime errors, e.g.
  - Opening a non-existent file
  - Invalid type cast
  - Accessing an index out of dimensions
- Usually causes program to halt
- If an exception occurs, a program should terminate elegantly
  - Error message is displayed
  - Exit or correct the error
  - Data integrity is guaranteed, e.g. close files
  - etc

Exceptions

- Many exceptions can be prevented by careful coding
  - Example: input validation
- Some exceptions cannot be avoided by careful coding
  - Example: failure of opening a file in read mode
- Control is passed from the point of error to an exception handler to deal with the error

Exception handler

- **Syntax:**

```
try:
    # statements that can potentially raise
    # an exception
except exceptionName:
    # statements to handle exceptionName
    # exception (exception handler)
```

Handling Multiple Exceptions

- Often code in the try clause can throw more than one type of exception
- Write except clause for each type of exception to be handled
  - Examples of *exceptionName*: IOError, IndexError, KeyError, TypeError, ValueError, RuntimeError, etc
- Or catch all exceptions by omitting the *exceptionName*

Example of Exception Handling

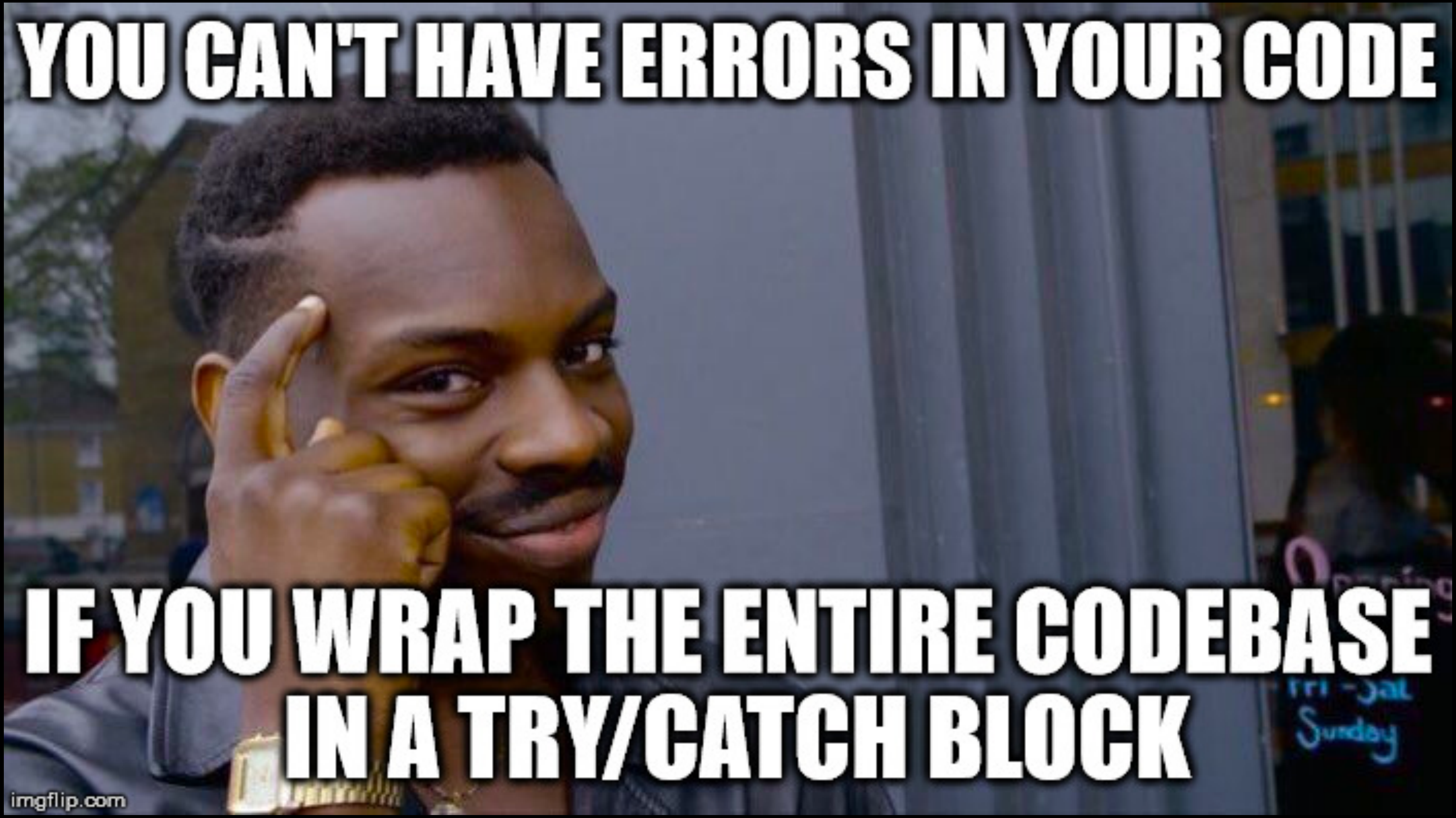
```
try:
    filename = input("Enter filename: ")
    in_file = open(filename, "r") # can raise a FileNotFoundError
    line = in_file.readline()
    value = int(line.strip()) # can raise a ValueError
except FileNotFoundError:
    print("Error: File not found")
except ValueError:
    print("Error: File contains invalid content")
except:
     print("Error")
```

If either of these exceptions is raised, the rest of the instructions in the try block are skipped!

What If an Exception Is Not Handled?

- Exceptions indicates that something exceptional (and bad) has happened
- The program halts or crashes
- Python documentation provides information about exceptions that can be raised by different functions:

(https://docs.python.org/3/c-api/exceptions.html?highlight=exceptions)


YOU CAN'T HAVE ERRORS IN YOUR CODE
IF YOU WRAP THE ENTIRE CODEBASE IN A TRY/CATCH BLOCK
imgflip.com

Try/except (ex_11.2.py)

- Download the file “ex_11.2_intnumbers.txt” from Brightspace
- Write a program to calculate the average of all numbers in the file
- The program should handle any IOError, ValueError, etc. and continue until the end of the file!
- Test the program
  - when the file does not exist
  - when a string, float or whitespace is read instead of an int