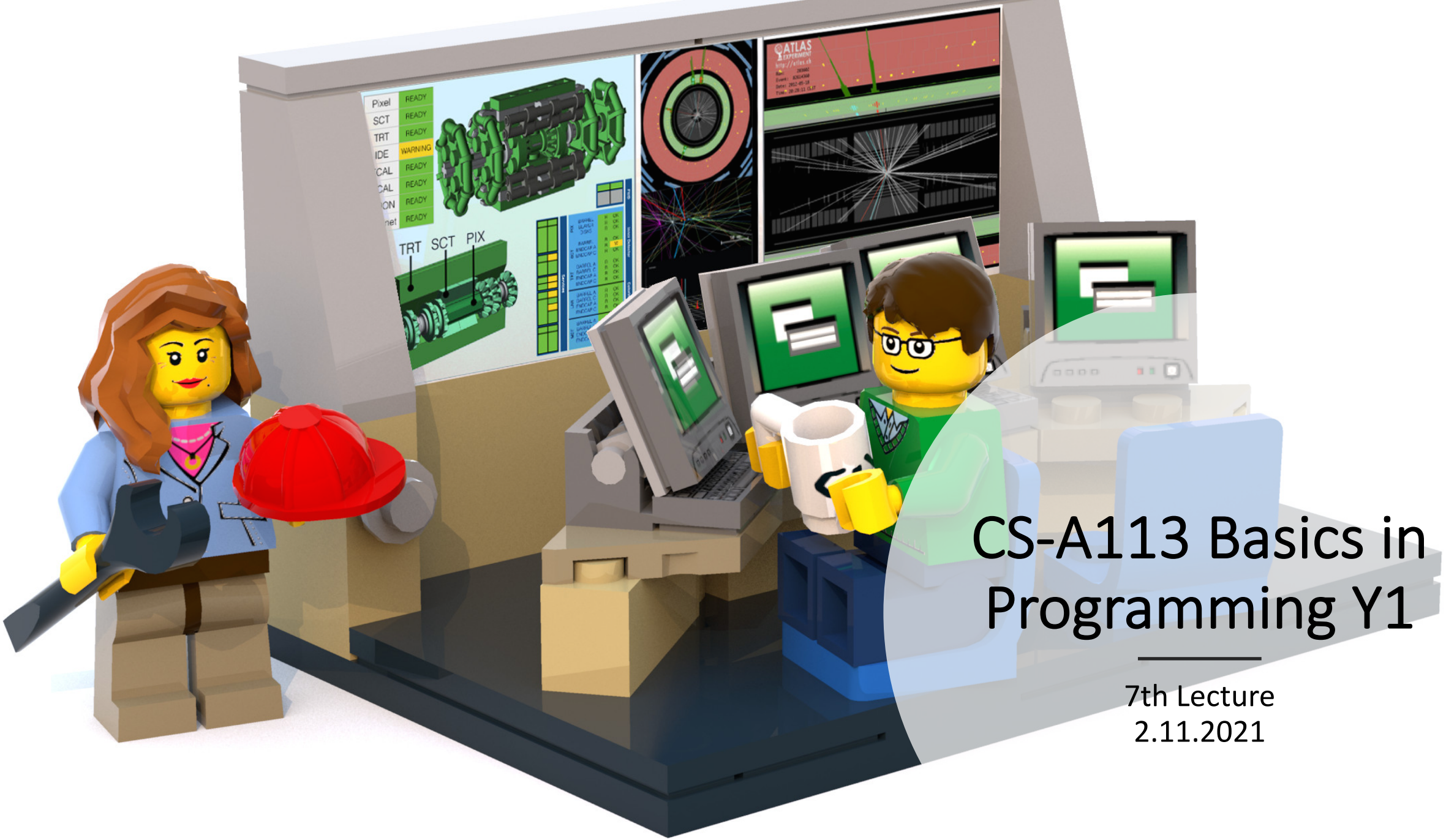


CS-A113 Basics in Programming Y1

7th Lecture
2.11.2021

Topics Today

Read Files

Exeption Handling

Programming so far

- A program is a series of commands that manipulate data
  - a = b * c
- Basic structures
  - Loop (for and while)
  - Branch (if, elif, and else)
- Data can be
  - Text or numbers
  - Single variables or structured (lists and dictionaries)
- Today we start to read data from files

Reading a File

What are Files

- Named data storage on the mass media
  - The disk, non-volatile memory
- Can have many different formats
  - File formats is a larger issue, which we do not discuss
- Today we assume files have lines of data
  - Text or numbers written as text, separated by newlines

How to Open and Close a File

Open the file

```
sourceFile = open(”text.txt”,”r”)

sourceLine = sourceFile.readline()

while sourceLine != ””:
    print(sourceLine)
    sourceLine = sourceFile.readline()

sourceFile.close()
```

Do some reading

Close the file

How to Read a File

```
sourceFile = open("text.txt","r")

# read line by line
sourceLine = sourceFile.readline()

while sourceLine != "":
    print(sourceLine)
    sourceLine = sourceFile.readline()

sourceFile.close()
```

```
sourceFile = open("text.txt","r")

# read through all lines
for sourceLine in sourceFile:
    print(sourceLine)

sourceFile.close()
```

```
sourceFile = open("text.txt","r")

# store all the lines in a list
lineList = sourceFile.readlines()

sourceFile.close()

for sourceLine in lineList:
    print(sourceLine)
```

Reading Files

- *open* tells the operating system to open the file for reading
  - Data from the beginning of the file is buffered into memory
  - OS knows that file is open
- *open* returns a file object
  - It knows how far the file has been read (position)
  - It can be asked for the next line
- After reading the file should be closed
  - Releases resources

Files are Objects

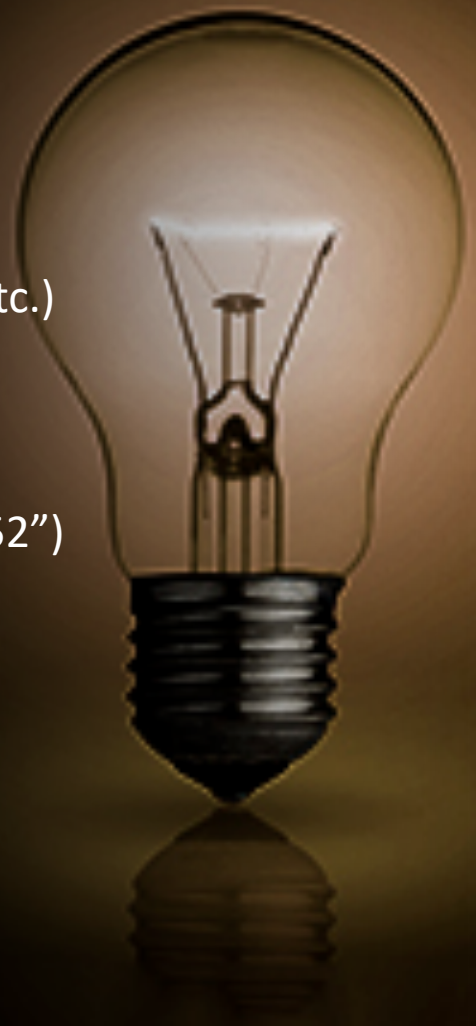
- *my_file = open("filename")*
  - *open* returns an object
- *line = my_file.readline()*
  - *readline* is a method of the my_file *object*
- *my_file.close()*
  - So is *close*
- The methods used to handle the file are connected to the file object
- We'll discus the object model a bit later on the course

Processing the File

- *readline()* returns a line of text as a *string* ending in the newline character \n
  - "line of text\n"
  - numbers are text, too
- Python has tools for manipulating strings
  - Methods of the str class

Good to Know

- line = line.rstrip()
  (removes whitespace characters at the end of the line, like newline, tab etc.)
  example: ”Hi, how are you? \n” → ”Hi, how are you?”
- parts = line.split(”,”)
  splits the string into several parts with ”,” as delimiter
  example: ”Barbara, Keller,123, ,52” → (”Barbara” , ” Keller” , ”123”, ”” , ”52”)
- myFile.readline() returns the empty string, once its finished
  example: ””
  keep in mind, an empty line is not the same: ”\n”



Example 1

```
def main1():

    myFile = open("lines.txt","r")
    myLine = myFile.readline()
    print(myLine)
    i = 0

    for theLine in myFile:
        print(theLine)
        if i==1:
            test=myFile.readline()
        i += 1

    myFile.close()
```

Content of lines.txt

```
Line0
Line1
Line2
Line3
Line4
```

Output

A: Line0, Line0, Line1, Line2, Line3, Line4

B: Line0, Line2, Line3, Line4

C: Line0, Line1, Line3, Line4

D: Line0, Line1, Line2, Line4

Example 2

```
def main1():

    myFile = open("lines.txt","r")
    myLine = myFile.readline()
    print(myLine)
    i = 0

    for theLine in myFile:
        print(theLine)
        if i==1:
            test=myFile.readlines()
        i += 1

    myFile.close()
```

Content of lines.txt

```
Line0
Line1
Line2
Line3
Line4
```

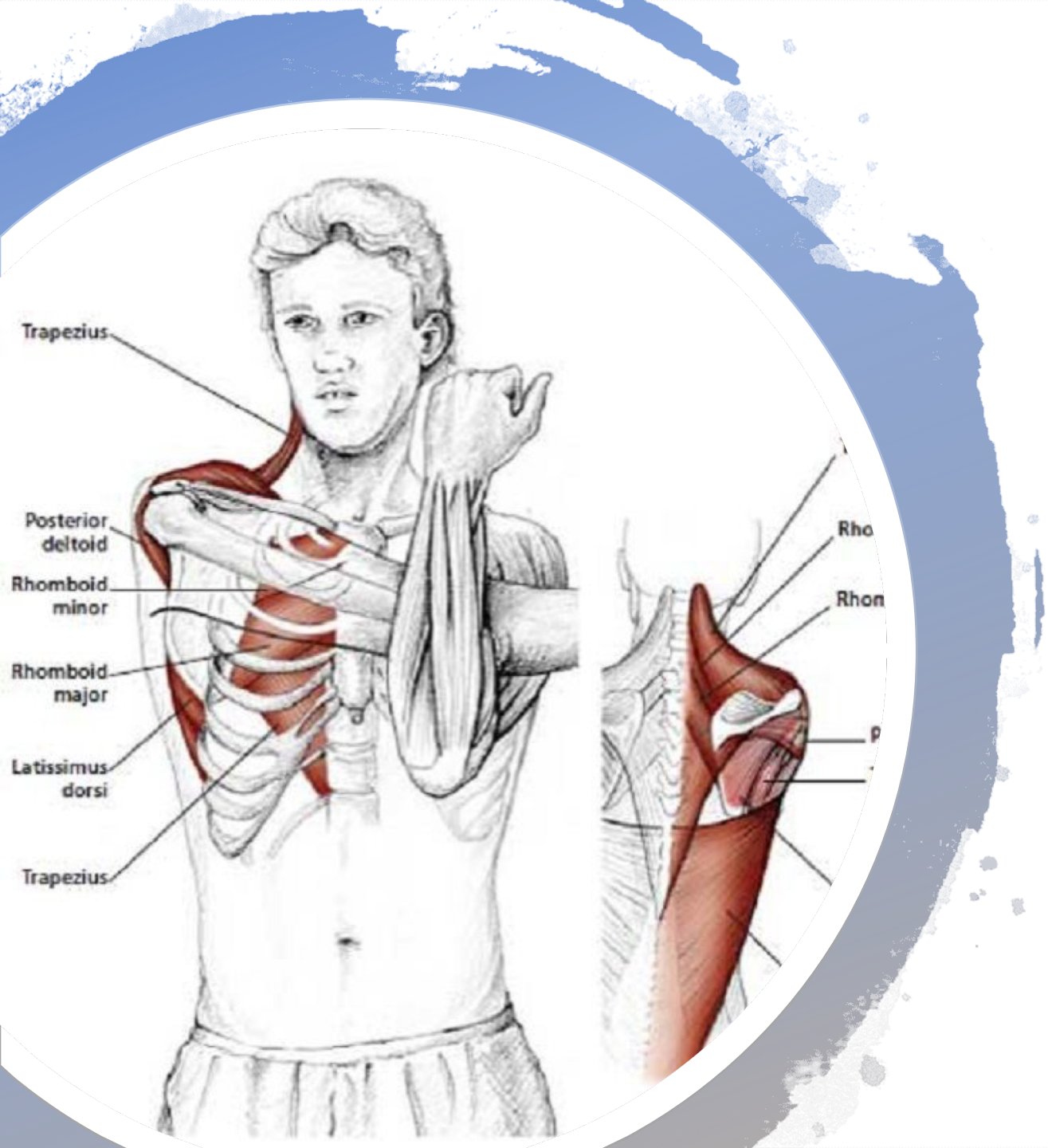
Output

A: Line0, Line0, Line1, Line2, Line3, Line4

B: Line0, Line1, Line2, Line3, Line4

C: Line0, Line1, Line2

D: Line0, Line1



Break: Move your Shoulders

Site Not Found

Well, this is awkward. The site you're looking for is not here.

Is this your site? Get more info or contact support.

DreamHost

What if the File you want to read does not exist in this directory?

Exception Handling

Exception Handling

```
try:
    #Here comes the code that maybe leads to an error
except ERROR:
    # What should you do in case of such an error
```

```
try:
    #Here comes the code that maybe leads to an error
    sourceFile = open(”text.txt”,”r”)
    sourceLine = sourceFile.readline()
except OSError:
    # What should you do in case of such an error
    print(”Error in reading the file.")
    print("Maybe the file is in another directory.”)
```

Example

```
def main1():
    name = input("Enter file name:")
    sum = 0
    count = 0

    try:
        tempfile = open(name, "r")
        for line in tempfile:
            parts = line.split(",")
            temperature = float(parts[1])
            sum += temperature
            count += 1
        tempfile.close()
    except OSError:
        print("Error in reading file ", name)
    except ValueError:
        print("Incorrect temperature in file ", name)
```

```
def main2():
    name=input("Enter file name:")
    sum = 0
    count = 0

    try:
        tempfile = open(name, "r")
        sum = 0
        count = 0
        for line in tempfile:
            parts = line.split(",")
            try:
                temperature = float(parts[1])
                sum += temperature
                count += 1
            except ValueError:
                print("Incorrect temperature in file ", name)
        tempfile.close()
    except OSError:
        print("Error in reading file ", name)
```

Exceptions

- Unexpected things can happen when running a program
  - The operating environment may throw surprises
    - Missing files
    - Errors in input data
  - These are called exceptions
- Exceptions can be caught and processed
  - Try-catch is usual name for this
  - In Python try-except

Exceptions in Python

- Suspectful part of code is executed in a *try:* block
  - Indentation marks the block
- After the block are *except:* statements that handle different cases
- Generally *try:* should be used sparingly
  - Large *try:* blocks become hard to understand
  - File operations are typical cases for using try
    - Open, read, convert data…

More about Errors and Exceptions

- Syntax errors are Python language errors
  - E.g., a === 1
  - Usually caught when the program starts
- Exceptions are various conditions that can be caught and resolved
  - E.g., int("five")
  - Python has dozens of specific exceptions
    - Out of memory, division by zero…
- Your program can also *raise* an exception
  - Way for a subroutine to tell the calling program that it can not perform


"That'
Folks!"
Kahoot!