Table of Contents:

[**Screenshots:**](#_kdroavbbs7r2) **2**

[**Basic summary of the initial code:**](#_pobwhb5933ne) **3**

[**Summary of the final code:**](#_z65l0cf7jcz) **4**

[**Bonus Question:**](#_fn0wxzkfbz2x) **6**

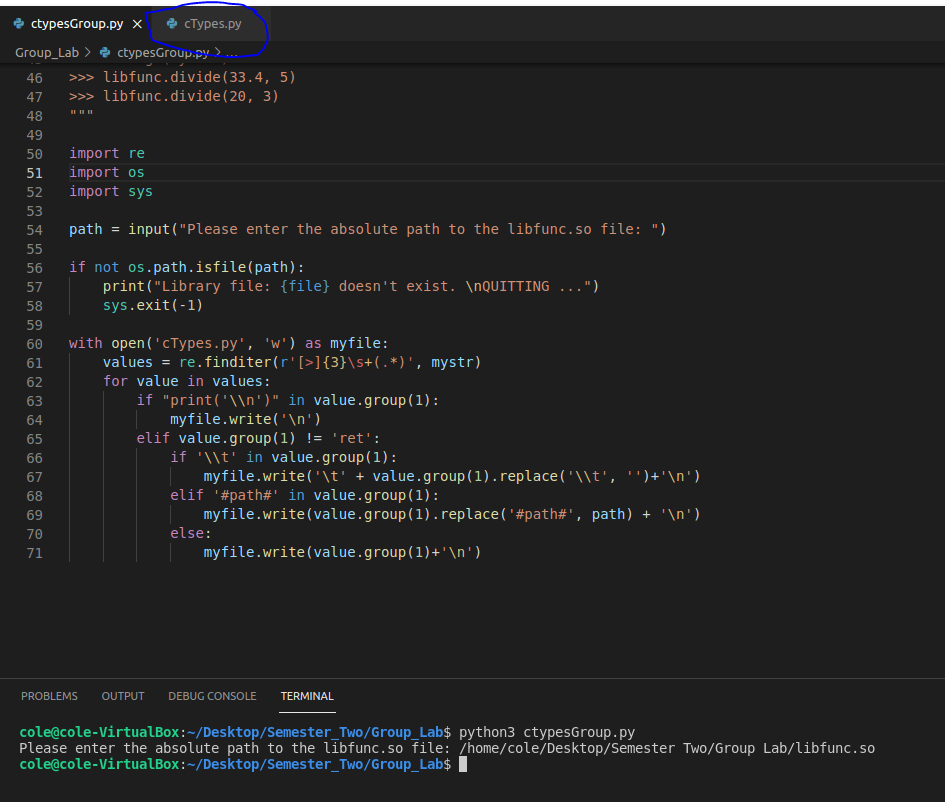
[**Resources:**](#_lc1ogh3ozehy) **7**

Assignment by: Cole Sanheim, Jack Chen

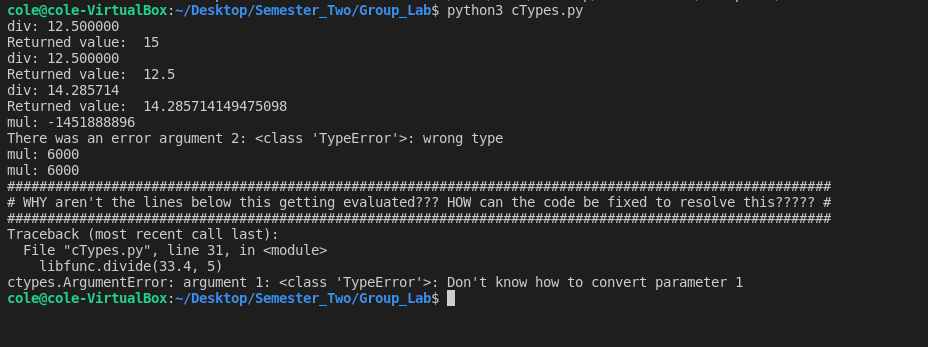
# 

# Screenshots:

This is the output of the original file, it creates another file called cTypes.py



This is the screenshot of the output of the final program



# Basic summary of the initial code:

The original code assigns a docstring to the variable **‘mystr’**, then imports the modules for **re**, **os**, and **sys**.

The input for the absolute path to the **libfunc.so** file is then asked. If the path is incorrect then the program will print **“Library file: {file} doesn’t exist. Quitting…”** and proceed to exit the program. When successful the code will assign the input to the variable path. The program will then open a file called **cTypes.py** for writing using the **with** context manager in python with variable **‘myfile’** as the name to reference for all further functions.

The variable **‘values’** is then created with ‘**re.finditer(r’[>]{3}\s+(.\*)’, mystr)’** assigned to it. The function parses and converts the contents of **mystr** that contain the same format as the ones specified in the **‘finditer’** function, which returns an **‘iterator’** containing the matched items that are not repeated, by scanning it from left to right and storing it in the order that it was found.

The rest of the code begins with a **‘for loop’** that is set to look at the **‘value’** within the variable **values** created by the previous line. The code then lists multiple conditional statements to accomplish different tasks depending on the content that is within the **‘value’** during the current run of the **‘loop’**. In other words, on **line 63** if the string **‘print(‘\\n’)’** is within the **‘value’**, the code will then execute **‘myfile.write(‘\n’)’** to the file that was opened previously. This idea is then repeated for the remaining conditional statements within the code.

# Summary of the final code:

The final code defines a function called **‘message’** that takes a string as an argument, it defines a variable **‘lent’** as the length of that string, it then prints out a number of **‘#’s** equal to **lent+4**, it then prints out the string preceded by and followed by a **‘#’** and then it prints out another line of **‘#’s**.

The main part of this code starts off by calling the **libfunc.so** file to be loaded by **CDLL.**

It then sets a variable called **‘ret’** to equal the return value of **libfunc.divide(100, 8)**, which presumably divides the first value by the second. It also prints out the result of the division which equals **12.5**.

Then there is a **print** statement printing out the value of **‘ret’** which prints **15** which is incorrect, we assume this is because the return value is an **‘INT’** and not a **FLOAT** or **DOUBLE**.

Next it sets the return type of the **divide** function to be of the data type **C\_FLOAT**

It then redos the function call of the **divide** function using the same values, which still outputs **12.5**. It also redos the **print** statement with the updated **‘ret’** value, which now equals **12.5** instead of **15**.

Next it calls the function **divide** again but this time it passes it the values **100** and **7**, which prints out **14.285714** as the result of the division of **100/7**.

It also does a **print** statement that prints out the return value of the function, which is **14.285714149475098**, which is the same value as before but with a higher accuracy.

In the next section it sets the expected argument types of the function **multiply** to be: **C\_INT, C\_FLOAT, C\_INT**.

It then calls the **‘multiply’** function and passes it **10**, **20** and **30**, this results in an output of -1451888896, which is \*very\* wrong, this is presumably because the **multiply** function cannot handle **FLOAT** values.

Next there is a **try**: statement which is trying to call the **multiply** function again and pass it 10, ‘aaa’ and 30.

The **except** statement catches this however, as it prints out an error ‘wrong type’ which is because it is expecting a **FLOAT** value but it was passed a string instead.

Then it sets the arguments of the **multiply** function to expect three **C\_INT**’s

It then calls the **multiply** function again with the same values (10, 20, 30) and it outputs the expected value: 6000.

Next it sets the return type to be that of **NONE**. (Note: this won't change the output as the output and the return values are different.)

It then calls the same **multiply** function as before and it still outputs 6000.

Next it sets a variable **‘mystr2’** to equal the string **"WHY aren't the lines below this getting evaluated??? HOW can the code be fixed to resolve this?????".**

It then calls the **message** function as described at the beginning with this string **‘mystr2’**, which results in the output: (this looks better in the terminal)

#######################################################################################################

# WHY aren't the lines below this getting evaluated??? HOW can the code be fixed to resolve this????? #

#######################################################################################################’

Finally it calls the function **divide** again and passes it the values 33.4 and 5. This results in an error occurring: types.ArgumentError: argument 1: <class 'TypeError'>: Don't know how to convert parameter 1. This appears to be because the function is expecting an argument of **C\_INT** type but instead we pass it a **FLOAT** value.

The last line calls the **divide** function one more time and passes it 20 and 3, but it never resolves as the code runs into an error on the line above.

# 

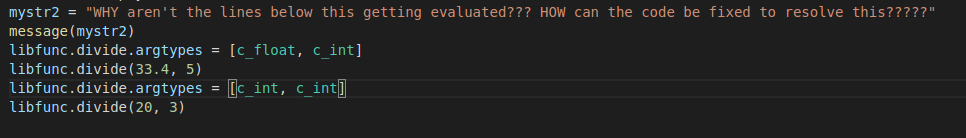
# Bonus Question:

If you add two lines to the code, one before the **divide** function call which causes the error, and one before the final **divide** function call, you can fix the error.

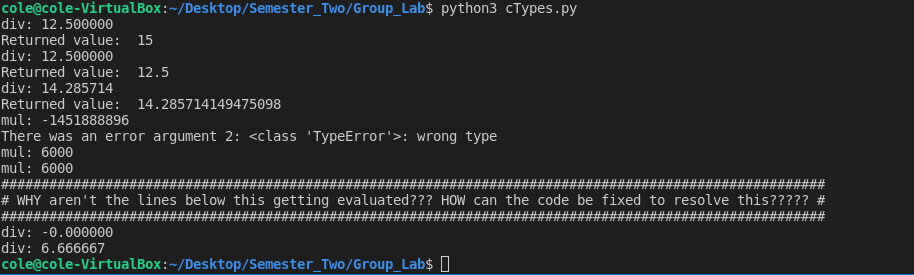
The first line to input sets the argument types for the **divide** function to be **C\_FLOAT** and **C\_INT**, this will no longer cause the error as it is now expecting the **FLOAT** value instead of an **INT** one.

The second line to add will be resetting the argument types back to **C\_INT** and **C\_INT** so it doesn't break on the final **divide** function call.

Updated code:



Updated output:



# 

# Resources:

[**https://docs.python.org/3/library/ctypes.html#module-ctypes**](https://docs.python.org/3/library/ctypes.html#module-ctypes)

[**https://docs.python.org/3/library/ctypes.html#specifying-the-required-argument-types-function-prototypes**](https://docs.python.org/3/library/ctypes.html#specifying-the-required-argument-types-function-prototypes)

[**https://docs.python.org/3/library/ctypes.html#type-conversions**](https://docs.python.org/3/library/ctypes.html#type-conversions)

[**https://docs.python.org/3/library/ctypes.html#return-types**](https://docs.python.org/3/library/ctypes.html#return-types)

[**https://docs.python.org/3/library/re.html**](https://docs.python.org/3/library/re.html)