Lab 8 – PART 1

Topics Covered:

- Exceptions

Pre-Lab Questions

Question 1: Select the appropriate error to be put in this except clause:

```
try:
   f = open("thefile.file","r")
except _______ as e:
   print("This file does not exist")
```

a) ZeroDivisionError
b) IOError
c) ValueError
d) SyntaxError

Question 2: Select the appropriate error to be put in this except clause:

```
for i in range(-3,7):
   try:
      print(7/i)
   except _________:
      print("On iteration",i," tried to divide by zero!")
```

a) ZeroDivisionError
b) IOError
c) ValueError
d) SyntaxError

Question 3: A try statement can have an 'except' AND a 'finally' clause. What does the finally clause do?

a) Allows you to execute code when an exception is thrown
b) Allows you to execute code when no exception is thrown
c) Allows you to execute code regardless of whether or not an exception is thrown

Activity:

Look at the following code segment. We have hardcoded some values for you in the list 'values' and created a loop through the values in the list. If the value is a string raise a ValueError that says "This is a string!". Use this code as a guide and fill in the yellow placeholder.

```
values =[1,2,3,4,5,"hello",6,7,8,9,"10"]
for cur in values:
    print("The value is :", values[cur])
    if type(values[cur]) == str:
        _____________________
```

Lab 8 – PART 2

1. **Review the examples of exceptions and then modify the program to include except clauses.**

   What is the difference between these 2 except clauses?

```
except ValueError:
    print("Value Error")
except ValueError as exception:
    print("Error:", str(exception))
```

   The first one prints the string we specified, which is "Value Error" when the body of this handler is executed, while the second one stores the exception object in a string and prints the message included with the exception, for example "invalid literal for int() with base 10: '35x2'".

   The finally clause is used when you need to take some action whether or not an exception is raised. A common example is using it to close files. It's important to always close a file that you open, but if you run into an error your program might terminate before you reach the

file.close() statement. The finally clause can be used in this situation because it will execute even if an exception was raised.

```
outfile = open(filename, "w")
try:
    writeData(outfile)
finally:
    outfile.close()
```

What is the difference between raise and except in python? When do I use except and when do I raise an exception?

- Think about it this way. When you use a Try-Except you are catching any errors that might occur and choosing how to handle them. On the other hand, when you raise an exception, it's because you are choosing to throw an exception if a condition occurs. For example, in this code segment, if you wanted to raise an exception and stop the program from running when x is lower than 0, you could do something like this:

```
x = -1
if x < 0:
    raise Exception("Sorry, no numbers below zero")
```

2. Copy the following functions into a new Python file.

```
def calc (div):
    try:
        q = 12 / div
        print(q)
    except:
        print("Cannot divide by 0!")

def main ():
    div = input("Enter a divisor: ")
    try:
        div = int(div)
        calc(div)
        print("Finished")
    except ValueError:
        print("Cannot convert to an int!")

main()
```

Examine this code to see what it is doing and think about the possible exceptions that could occur. What different user input values would produce which type of exceptions? How would each one be handled? Think about these questions before running the code and trying it out.

Now run the code several times and see the output when you try entering each of the following input values when prompted for user input: **3**, **6**, **0**, **abc**

For each one, look at the output and think about this is happening. Which exceptions are being handled inside the main() function and which are being handled inside the calc() function. Notice that "Finished" is not printed out when we type in non-numeric values. This is because that exception is not handled yet at the time it would be executing that line so execution jumps straight down to the "except ValueError:" line and does not finish running the code above it. However, if we type in 0, we do see "Finished" printed at the end because that exception was already handled in the calc() function.

Activity:

The following code segment is a short program that has some lines that may raise an exception. Find these lines, and come up with 2 except clauses in a Try-Except to handle these exceptions and modify the program.

```
filename = input("Enter filename: ")
infile = open(filename, "r")
line = infile.readline()
value = int(line)
```