Preventing Run-Time Errors Using Exception Handling

When a run-time error occurs in Python, an exception is thrown which basically means that Python will stop the normal execution of the program and most often return an error message.

Errors Resulting from Incorrect User Input

One common run-time error results from incorrect user input. For example, given the following code:

```
import math # Imports the math module
numForMath = int (input ("Enter an integer: ")) # Stores a number for a math operation
answer = 20 / numForMath + math.sqrt (numForMath)
```

Assume the user inputs a 0. The program will crash and the following error message will appear:

ZeroDivisionError: int division or modulo by zero.

Assume the user inputs a string like “one”. The program will crash and the following error message will appear:

ValueError: invalid literal for int() with base 10: 'r'

Assume the user inputs a 6.7. The program will crash and the following error message will appear:

ValueError: invalid literal for int() with base 10: '6.7'

Assume the user inputs a -4. The program will crash and the following error message will appear:

ValueError: math domain error

Errors With Files

Another common type of error results from files which cannot be opened correctly. For example, given the program has the following code:

```
fileName = input (“Enter the name of the file”)
snIn = open (fileName, “r”)
```

Assume the user typed in the name data.dat and there was no such file. The following error message will appear:

IOError: [Errno 2] No such file or directory: 'data.dat'

Out of Range Errors

Finally, another common run-time error occurs when the user is asked to output an element in a list which doesn’t exist. For example, assume the following code is in a program:

```
x = [1,2,3] # Initializes a three element list
subscript = int (input ("Which element"))
print (x[subscript])
```

Assume the user typed in5 and there was no such element. The following error message will appear:

IndexError: list index out of range

Dealing With Errors Using Exception Handling

The good programmer will not allow the program to crash. Instead, they will allow the program handle the errors gracefully by allowing the user to correct the error or at least giving them a message to let them know why their program does not work.

There are a couple of ways of doing this but **exception handling** is commonly used. The theory behind exception handling is that the program is coded to try to execute a line of code. If it works, great! If there is an exception, the program will handle it (ie. If it doesn't work, the program will deal with it instead of crashing). The code to do this is:

```
import math    # Imports the math module

while True:    ''' Replaces the use of the bogus variable. Assumes that there is a boolean variable that
               is always True Will continue to loop until a break is encountered'''

    try:       # The code where the exception could occur is attempted.
               numForMath = int (input ("Enter an integer: ")) # Stores a number for a math operation
                answer = 20 / numForMath + math.sqrt (numForMath)

    except ZeroDivisionError :    # Message outputted if a  ZeroDivisionError is encountered.
               print ("Can't divide by 0.  Please re-enter the integer.")

    else:      # Loop is exited if no exceptions are encountered
               break

print ("The answer to the equation  20 / inputted number  + the sqrt of the inputted number is",
answer)
```

Assignment:

Save the following programs in a folder called **Exception Handling**.

1. a) Type the small program above to ensure that it works. Note that it will handle the run-time error encountered if 0 is inputted. Call your program **ExceptionOne.py**

   b) Improve the program above to deal with errors encountered by other incorrect values inputted by the user (ie. "One", -2, 6.7). These are ValueErrors. Re-save Call your program **ExceptionOne.py**
   Hint: Add the following after the current except command and add an appropriate error message.

   ```
   except ValueError :    # Message outputted if a Value Error is encountered.
   ```

2. Program Files6.py required the user to input the name of the file. Update your code to insure that the program does not crash if the user inputs an incorrect filename. (**Hint**: Look at the notes above to figure out which exception to use). Save your updated code as **ExceptionTwo.py**

3. Program List4.py required the user to input the contents of any element as one of the options. Update your code to insure that the program does not crash if the user inputs an inappropriate subscript. (**Hint**: Look at the notes above to figure out which exception to use). Save your updated code as **ExceptionThree.py**