PRACTICE WITH EXCEPTIONS

1. The following line of code might raise a `ValueError`:

   ```
   def inputInteger():
       n = int(input("Enter an integer: "))
       return n
   ```

   Call this function from within a `try` statement and use `except` to catch any `ValueError` that occurs. If a `ValueError` occurs, then display a helpful error message.

2. Modify your previous program so that, if the user does not enter an integer, the program asks the user for another integer. The program should continue asking until the user enters an integer.

3. Write a function that intentionally raises an `IndexError` exception when called. Then call your function from within a `try` statement and use `except` to catch the error.

4. **Bonus:** The following code retrieves a file from a web server:

   ```
   import urllib.request
   result = urllib.request.urlopen( someURL )
   ```

   However, the function `urllib.request.urlopen()` can raise various exceptions. For example, it raises a `ValueError` if it cannot interpret its argument as a valid URL. It also raises a `urllib.error.HTTPError` if the server cannot find the requested file. To see this for yourself, run the following code:

   ```
   import urllib.request
   urllib.request.urlopen( "test" )  # this line raises a ValueError
   urllib.request.urlopen( "http://www.mlwright.org/nofile.txt" )
      # this line above raises a urllib.error.HTTPError
   ```

   Write a program that asks the user for a URL and then uses `urllib.request.urlopen()` to request that URL. If `urllib.request.urlopen()` raises a `ValueError` or `urllib.error.HTTPError`, then your program should handle the exception and print a helpful message for the user.

   If no error occurs, you can (usually) print the HTML file returned like this:

   ```
   byteData = result.read()
   strData = byteData.decode("utf8")
   print(strData)
   ```

   Once you have the data from the file stored in a Python string, you can extract information from it. Experiment with using Python to read web pages!