Exceptions with User Input

Exceptions can occur for reasons that do not involve file-handling. Typecasting user input as an integer by nesting an `input` function inside of a call to `int` (like below) is dangerous because if the user enters invalid data (any non-numeric character, other than '-'), the program will crash.

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

To prevent an exception being thrown, it is better to try to convert the data to an integer and, if the attempt fails, use a loop to have the user re-enter the data. The example below does not crash when bad data is entered.

The table below summarizes some of the more common exceptions, but there are many more. Information can be found in the Python documentation's Built-In Exceptions page.

Example of Handling Exceptions Related to User Input

```
while True:
      n = input("Enter an integer (enter a blank value to stop): ")
      if n == '':
            break
      try:
            n = int(n)
      except ValueError:
            print("Invalid value, must be an integer.")
      else:
            print("You entered the value", n)
```

Common Built-In Exceptions

Exception	Description
`TypeError`	An action is attempted on an object that is an inappropriate data type (e.g. passing a string to a function, when an integer is expected).
`ValueError`	A action is attempted on an object that is an appropriate data type, but the value is inappropriate (e.g. attempting to convert non-numeric values to an integer).
`ArithmeticError`	A mathematical issue occurs due to a numeric calculation. Includes `ZeroDivisionError`, `FloatingPointError`, and `OverflowError`.
`ZeroDivisionError`	Occurs when one value is divided by zero (or modulo zero). A special case of `ArithmeticError`.
`OSError`	Various Operating System errors (e.g. a file is not found, there is no disk space to write to a file, the user does not have appropriate permissions, and so on). May appear as `IOError` in older Python documentation.
`FileNotFoundError`	A specified file cannot be found. A special case of `OSError`.

Exceptions with User Input

Answer the following questions.

1. What type of exception is thrown when each snippet of code below is executed?

```
n = 3
s = “hello”
print(n+s)
```

```
import math
x = math.sqrt(-5)
print(x)
```

```
p = 3.14159
q = p * 10**1000
print(q)
```

2. Is it necessary to write code to deal with *every* possible exception? Explain why or why not.

3. Make a list of the exceptions you would expect to occur in most programs, and give an example of how they might occur.

Write programs that accomplish each task. Use proper conventions for variable names, input prompts, output statements, and program structure. Do *not* assume that the user will enter the correct data type or value for any input, or that files contain the correct data types or values.

4. Write the following functions, and save them in a module called `userinput.py`. Use this module to obtain user input in future programs, so that you do not have to rewrite the code each time.

   - `GetInt(p)`: obtains an integer from the user, with prompt *p*.
   - `GetPosInt(p)`: obtains a positive integer from the user, with prompt *p*.
   - `GetFloat(p)`: obtains a floating-point value from the user, with prompt *p*.
   - `MakeIntList(L)`: given a list *L*, generate a (possibly empty) list containing all values that are, or can be converted into, integers. Note: if value is a string (e.g. “2.72”), then `int(x)` will throw an exception, but if value is a float, `int(x)` will round it down to the nearest integer (e.g. 3.14 will become 3, which is probably not desired). There are many ways around this, using the `is_integer()` method, the built-in function `isinstance(value, type)`, using modulus, and so on. Your code should handle both cases, whichever method you choose.

5. Read a string from the user, then ask the user to enter the index of the character s/he wishes to display. Catch any `IndexError` exceptions that occur, and redirect the user to enter a valid index.

6. When Python attempts to write to a file for which it does not have proper permissions (e.g. it is owned by another user, or it is a read-only file), it may throw a `PermissionError` exception. Create a file, and mark it as read-only, then attempt to open the file and write to it. Write an exception handler that will output a message instead of crashing.