Chapter 6: Model Solutions

Below, you'll find sample solutions to the lab exercises in the book.

Lab Exercises 6.1

In this solution, we use a try-except block to handle potential errors during user input. We attempt to convert the user's input for length and width into float values using float(input()). If the user enters a non-numeric value, a ValueError is raised. We also check if the length or width is negative, and if so, we raise a ValueError to handle that case.

```
try:
    length = float(input("Enter length of the rectangle: "))
    width = float(input("Enter width of the rectangle: "))

    if length < 0 or width < 0:
        raise ValueError("Length and width must be positive.")
    area = length * width
    print("The area of the rectangle is:", area)

except ValueError as ve:
    print("Error:", str(ve))
except Exception as e:
    print("An error occurred:", str(e))
```

Lab Exercises 6.2

1. Exception handling is a mechanism in programming that allows us to handle and manage errors or exceptional events that may occur during the execution of a program. It helps prevent program crashes and provides a way to gracefully handle unexpected situations. Exception handling is important because it improves program robustness, maintains program flow, and allows for appropriate error handling and recovery.

2. Opening and reading files: Exception handling can be used to handle file-related errors such as file not found, permission issues, or unexpected file content.

   User input validation: Exception handling can handle invalid user input, such as non-numeric values or out-of-range inputs, and provide appropriate error messages.

   Network operations: Exception handling can handle network-related errors, such as connection failures or timeouts.

   Database operations: Exception handling can handle database-related errors, such as connection errors or query failures.

3. Exception handling in Python is implemented using the try-except statement. The code that may raise an exception is placed inside the try block, and the code to handle the exception is placed inside the except block. If an exception occurs in the try block, the corresponding except block is executed.

4. The try block is used to enclose the code that may raise an exception. It allows us to identify the specific portion of code that may cause an exception. If an exception occurs within the try block, the code execution is immediately transferred to the corresponding except block.

5. You can catch and handle specific exceptions using the except block. You can specify the type of exception you want to handle after the except keyword.

6. In this solution, we use a while loop to repeatedly ask for user input until a valid number is entered. The float() function is used to convert the user's input into a floating-point number. If the conversion raises a ValueError, we display an error message and the loop continues.

```
while True:
    try:
        number = float(input("Enter a number: "))
        break  # Exit the loop if a valid number is entered
    except ValueError:
        print("Invalid input. Please enter a valid number.")

square = number ** 2
print("The square of", number, "is", square)
```

7. In this solution, we use the open() function to open the file in "read" mode and the with statement to automatically close the file after reading. If the file is successfully opened, we read its content and process it as required.

```
filename = "data.txt"

try:
    with open(filename, "r") as file:
        data = file.read()
    # Process the data as required
    print("File content:", data)
except FileNotFoundError:
    print("File not found:", filename)
except PermissionError:
    print("Permission denied. Cannot read file:", filename)
except Exception as e:
    print("An error occurred while reading file:", str(e))
```