Chapter 5: Model Solutions

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

# Lab Exercises 5.1

1. Calculate average

def calculate\_average(numbers):

if len(numbers) == 0:

return 0 # Return 0 if the list is empty

total = sum(numbers)

average = total / len(numbers)

return average

numbers = [5, 10, 15, 20]

average = calculate\_average(numbers)

print("The average is:", average)

# Lab Exercises 5.2

1. Calculate bill

def calculate\_bill(base\_price, tax\_rate, discount\_percentage):

tax\_amount = base\_price \* tax\_rate

discount\_amount = base\_price \* discount\_percentage

total\_amount = base\_price + tax\_amount - discount\_amount

return total\_amount

base\_price = 100

tax\_rate = 0.08

discount\_percentage = 0.2

total\_bill = calculate\_bill(base\_price, tax\_rate,   
 discount\_percentage)

print("Total Bill:", total\_bill)

# Lab Exercises 5.3

1. Calculate power

def calculate\_power(base, exponent=2):

return base \*\* exponent

# Test the function

print(calculate\_power(2)) # Output: 4 (square of 2)

print(calculate\_power(3, 3)) # Output: 27 (3 raised to the power of 3)

2. Function called `print\_person\_details`

def print\_person\_details(name, age, country):

print(f"Name: {name}, Age: {age}, Country: {country}")

# Test the function

print\_person\_details("John", 25, "USA")

print\_person\_details("Emma", 30, "Canada")

3. Recursive function called `fibonacci()`

def fibonacci(n):

if n <= 1: # Base case: Return [0] for n <= 1

return [0]

elif n == 2: # Base case: Return [0, 1] for n == 2

return [0, 1]

else:

fib\_seq = fibonacci(n - 1) # Recursive call

# Append the sum of the last two numbers in the sequence

fib\_seq.append(fib\_seq[-1] + fib\_seq[-2])

return fib\_seq # Return the updated Fibonacci sequence

# Print the Fibonacci sequence up to the 10th term

print(fibonacci(10))

4. A parameter is a variable defined in the function declaration. It represents a value that the function expects to receive when it is called.

An argument is a value or expression passed to a function when it is called. Arguments are assigned to the corresponding parameters inside the function.

5. A built-in function is a function that is provided as part of the Python programming language. These functions are readily available and can be used without requiring any additional imports or installations. Examples of built-in functions include `print()`, `len()`, `type()`, `range()`, etc.

6. A user-defined function is a function created by the programmer to perform a specific task or set of operations. These functions are defined using the `def` keyword and can be customized to suit the requirements of the program. User-defined functions are created to encapsulate reusable pieces of code and promote code modularity and reusability.

7. A function is considered recursive if it calls itself during its execution. Recursion involves solving a problem by breaking it down into smaller, simpler instances of the same problem until a base case is reached. The base case is a condition that stops the recursion and provides the result or termination point. Recursive functions are useful for solving problems that can be divided into subproblems that are identical in nature but smaller in size.

8. A local variable is a variable that is declared inside a function and can only be accessed within that function. It has a limited scope and exists only for the duration of the function's execution. Local variables are typically used for temporary storage or intermediate calculations within the function.

A global variable, is a variable that is defined outside of any function and can be accessed from anywhere within the program, including inside functions. It has a global scope and exists throughout the entire program's execution. Global variables are useful for storing data that needs to be accessed or modified by multiple parts of the program. However, it is generally recommended to minimize the use of global variables to maintain code clarity and avoid unintended side effects.