Assignment 3 Shortcuts, Command Line, and Packages

Subject: Computer Science Workshop - 1 (CSE 2141)
Session: September 2025 to January 2026
Branch: Computer Science and Engineering (CSE)
Section: All
Course Outcomes: CO1, CO2
Program Outcomes: PO1, PO2, PO3, and PO5

Learning Levels: Remembering (L1), Understanding (L2), Application (L3).

Q no.	Questions	Learning Levels

Q1.	Write a Python program that demonstrates the use of **multiple assignment** and **tuple unpacking shortcuts**. The program should perform the following tasks:
	• Prompt the user to enter three numbers separated by spaces.
	• Use a single-line assignment to store each number in its own variable (x, y, z).
	• Swap the values of y and z in one line using **tuple unpacking**.
	• Print the values of x, y, and z before and after the swap to show the effect of the shortcut.
	Example Output:
	`User input: 5 6 7`
	`Before swapping: x = 5, y = 6, z = 7`
	`After swapping: x = 5, y = 7, z = 6`
		L1, L2

Q2.	Write a Python program that simulates a **basic command-line calculator** supporting multiple operations (addition, subtraction, multiplication, and division). The program should perform the following tasks:
	• Accept user inputs in the format: {operation num1 num2} (for example, `add 5 3`).
	• Support the following operations: `add`, `sub`, `mul`, `div`.
	• Provide an option to exit the program with the command `exit`.
	• Print the result of each calculation immediately after execution.
	Example Output:
	`Enter operation (add/sub/mul/div) or 'exit' to quit:`
	`add 5 3`
	`Result: 8`
	`div 10 0`
	`Error: Division by zero is not allowed.`
	`mul 4 6`
	`Result: 24`
	`exit`
	`Program terminated.`
		L2, L3

Q3.	Create a Python program that demonstrates **advanced tuple assignment** with multiple features. The program should perform the following tasks:
	• Prompt the user to enter a list of integers separated by spaces.
	• Use tuple unpacking to extract the first two numbers as individual variables and the rest into a list using the **starred expression**.
	• Swap the first two numbers using **tuple assignment**.
	• Compute and print the sum of the remaining numbers.
	• Demonstrate unpacking where the starred variable appears in the middle (e.g., first, middle, last) and display the values assigned.
	Example Output:
	`Enter integers separated by spaces: 10 20 30 40 50`
	`First number: 10`
	`Second number: 20`
	`Remaining numbers: [30, 40, 50]`
	`After swapping: First: 20, Second: 10`
	`Sum of remaining numbers: 120`
	
	`Unpacking Example:`
	`First: 10`
	`Middle: [20, 30, 40]`
	`Last: 50`
		L1, L2

Q4.	Write a Python function that can accept any number of sales amounts as **positional arguments** (`*args`) and additional information (such as the name of the salesperson, date, and location) as **keyword arguments** (`**kwargs`). The function should perform the following tasks:
	• Add up all the sales amounts provided through `*args`.
	• Count and display how many pieces of extra information were provided through `**kwargs`.
	Example Output:
	`Total Sales Amount: 12500`
	`Number of Extra Information Items: 3`
	
	`Extra Information Provided:`
	`Name: John Doe`
	`Date: 2025-11-01`
	`Location: Bhubaneswar`
		L1, L2

Q5.	Create a Python decorator called `timer` that helps measure how long a function takes to run. • A **decorator** is a special tool that wraps around a function to add extra features without changing the original function's code. • The `timer` decorator should work with any function, regardless of the number of input arguments. • When a function with this decorator is called, it should: – Record the time before the function starts. – Execute the original function. – Record the time after it finishes. – Calculate and print how many seconds the function took to run. • To demonstrate its functionality, write a simple function that waits for a random duration (between 0.5 and 1.5 seconds), decorate it with `@timer`, and then call it. • The program should print both the sleep duration and the total time taken for the function to execute.	L2, L3

Q6.	Write a Python generator function called `filter_high_sales` that takes a list of daily sales amounts and a threshold value.
	• The function should **yield** only those sales amounts that are greater than or equal to the given threshold.
	• Demonstrate this generator by providing it with a list of sales and printing all high sales (e.g., sales above $500).
	Example Output:
	`Daily Sales: [250, 800, 450, 1200, 600]`
	`Threshold: 500`
	`High Sales:`
	`800`
	`1200`
	`600`
		L2, L3

Q7.	Write a Python generator function called `sensor_data_stream` that simulates incoming temperature readings from a sensor. • The generator should **yield** a new random temperature value between `20°C` and `30°C` each time it is called. • Demonstrate how to use this generator to process and print the first **10 sensor readings**. This exercise will help you understand how generators can be used to simulate real-time data streams and manage continuous data flow efficiently.	L2, L3

Q8.	Write a Python script that takes several numbers as **command-line arguments** and prints their sum.
	• Use `sys.argv` to access the numbers from the command line.
	• Ensure the script ignores the filename while calculating the sum.
	• Display a clear **error message** if any argument is not a valid number.
	• Show a sample command-line usage and the expected output.
	Example Output:
	`Command:`
	`python sumargs.py 10 20 30 40`
	
	`Output:`
	`Sum of numbers: 100`
	
	`Command:`
	`python sumargs.py 10 20 abc 30`
	`Error: Invalid input 'abc'. Please enter only numbers.`
		L2, L3