These exercises are designed to leverage the specific features of Bash (like piping and command line tools) and Python (like robust data structures and libraries), forcing you to address the same problem using two distinct paradigms.

The exercises below cover all topics from the lessons: **Redirection, Piping, Variables, Arrays, Conditional Statements, and Loops.**

Dual-Implementation Exercises (Bash & Python)

1. File Type Checker

Concept Covered	**Bash: Command-line Arguments ($1), Conditional Statements (if/elif/else), File Test Operators (-f, -d).** **Python: Conditional Statements, os.path module (e.g., isdir, isfile).**
Description	Create a script that takes a single filename/path as a command-line argument. It must report whether the path points to a **regular file**, a **directory**, or if it **does not exist**.
Bash Implementation	Use if, elif, and the Bash test operators ([-f "$1"], [-d "$1"]).
Python Implementation	Use if, elif, and functions from the os.path library (e.g., os.path.isdir(path)).

2. Array Filtering: Even Numbers

Concept Covered	**Bash: Array definition, Looping (for loop), Arithmetic Conditional Operators (-eq, %). Python: List definition, Looping (for loop), List indexing, Modulo operator (%).**

Description	Define an array/list of integers (e.g., [5, 12, 17, 20, 2, 9]). Iterate through the collection and print only the numbers that are **even**.
Bash Implementation	Use a for loop over the array elements (${array[@]}). Inside the loop, use the modulo operator (%) within an if condition.
Python Implementation	Use a for loop over the list. Use the modulo operator (%) within an if condition.

3. Basic Command-Line Calculator

Concept Covered	**Bash: Command-line Arguments ($1, $2, $3), Conditional Statements (case), Arithmetic operations ($(())). Python: Command-line arguments (sys.argv), Conditional Statements (if/elif/else), Type casting (int()).**
Description	Write a script that accepts three arguments: **Number 1**, **Operator** (+, -, x, /), and **Number 2**. It must perform and print the calculation based on the operator.
Bash Implementation	Use a case statement to handle the four different operators. Perform the calculation using Bash arithmetic expansion.
Python Implementation	Use if/elif to handle the four operators. Remember to convert the input arguments from strings to numbers before calculating.

4. Simple Countdown Timer

Concept Covered	**Bash: Looping (while loop, until loop), Variable decrement (let or $(())), External command (sleep). Python:**

	Looping (while loop), Variable decrement, time.sleep().
Description	Take a starting number (e.g., 5) as a command-line argument and print a countdown from that number to 1. The script should pause for 1 second between each number.
Bash Implementation	Use a while or until loop (as covered in Lesson 6). Decrement the counter and use the sleep 1 command.
Python Implementation	Use a while loop. Decrement the counter and use time.sleep(1).

5. Function to Check Exit Status

Concept Covered	**Bash: Functions, Special variable ($?), Conditional Statements (if), Exit status. Python: Functions, subprocess module, Error Handling (try/except).**
Description	Create a function that runs a given command and then prints a message indicating whether the command succeeded (exit status 0) or failed (exit status non-zero).
Bash Implementation	Define a function. Run a command inside the function (e.g., ls /etc/passwd or ls /nonexistent). Immediately check the value of $?.
Python Implementation	Use the subprocess.run() function. Check the returncode attribute of the result object.

6. User Input Array Population

Concept Covered	**Bash: Looping (for or while), Reading input (read), Array appending (arr+=(value)). Python: Looping, Reading input (input()), List methods (.append()).**
Description	Prompt the user to enter 5 names. Store these names in an array/list. After the loop, print all the names in a single line.
Bash Implementation	Use a for loop (from 1 to 5). Use read to get the input. Append the input to the array using the arr+=(value) syntax.
Python Implementation	Use a for loop. Use input() to get the name and the .append() method to add it to a list.

7. System Environment Variables

Concept Covered	**Bash: Variables ($VARNAME), Redirection (>), External command (env or printenv). Python: os module (os.environ).**
Description	Print the values of two specific environment variables (HOME and USER for example) and then redirect the output of **all** environment variables to a file named env_data.txt.
Bash Implementation	Use echo $HOME and echo $USER. Use the env command and the **redirection operator** (>) to save the full list to the file.
Python Implementation	Import the os module. Access the specific

	variables using os.environ['VARNAME']. Use file I/O operations (with open(...)) to write the full os.environ contents to the file.

8. Fibonacci Sequence Generator

Concept Covered	**Bash: Loops (while or for), Variables, Arithmetic calculations, Conditional termination. Python: Loops, Variables, Sequence generation.**
Description	Generate and print the first **N** numbers of the Fibonacci sequence (where N is a user-provided integer argument). The sequence starts with 0 and 1.
Bash Implementation	Use a loop with three variables to track the sequence: current, previous, and next. Use Bash arithmetic expansion ($(()) or let) for the calculations.
Python Implementation	Use a for or while loop. Use three variables to track the sequence and perform the addition.

9. String Reversal Function

Concept Covered	**Bash: Functions, Command-line argument passing to functions, External command (rev). Python: Functions, String slicing or iteration.**
Description	Implement a function that accepts a string as an argument and prints the reversed version of that string.
Bash Implementation	Define a function that accepts an argument ($1). Use the external command

	rev and **pipe** the input string to it, or iterate through the string characters using a loop (a more complex approach for learning).
Python Implementation	Define a function that accepts a string parameter. Use Python's string slicing feature ([::-1]) or a for loop to build the reversed string.

10. Log File Statistics Generator

Concept Covered	**Bash: Piping, awk/grep/sort/uniq, Redirection, Function. Python: File I/O, Dictionary/List, String splitting, Aggregation logic.**
Description	Assume you have a log file (or create a simple one with 10 lines of simulated IP addresses, status codes, and user IDs). The script must read the log and generate two statistics: 1) The total number of entries. 2) A list of unique user IDs/IPs found, along with a count of how many times each one appears.
Bash Implementation	Define a function. Use cat and **piping** with awk or a combination of grep, sort, and uniq -c to count unique occurrences. Redirect the final output to a file named report.txt.
Python Implementation	Use a dictionary to store the counts (IP/UserID as key, count as value). Iterate over the file lines, split the string, and update the dictionary. Write the final dictionary contents to report.txt.

11. Recursive File Extension Converter

Concept Covered	**Bash: Looping over command output (find), Arrays, Variable substitution/manipulation (filename modification), System command (mv). Python: os.walk, String methods (e.g., .replace()), Conditional logic.**
Description	Write a script that takes a starting directory path and an old file extension (e.g., .tmp) as arguments. It must recursively find all files with that extension and rename them to use a new extension (e.g., .log). The script should only rename files that exist.
Bash Implementation	Use find to locate the files, then pipe the result into a while read loop. Use parameter expansion (string manipulation) to construct the new filename and execute the mv command inside the loop.
Python Implementation	Use os.walk to traverse the directory tree. Use an if condition to check the filename extension and then use os.rename to perform the modification.

12. Array Slice and Merge with Function

Concept Covered	**Bash: Functions, Arrays, Array slicing (${arr[@]:s:n}), Function parameter passing (implicit array passing). Python: Functions, Lists, List slicing, Function parameter passing.**
Description	Define two separate arrays/lists of numbers. Write a function that accepts both arrays/lists as input. The function must: 1) Extract the first three elements from the first array/list. 2) Extract the last two elements from the second array/list. 3)

	Return/print a new array/list that is the merger of these two slices.
Bash Implementation	This is complex in Bash. The function must accept the entire array as arguments, which usually requires careful handling of spaces. Use **array slicing** within the function and a subshell command substitution to merge the results into a new array.
Python Implementation	Define a function that takes two lists. Use standard list slicing (list[:3], list[-2:]) and the + operator or .extend() method for merging.

13. Disk Usage Report with Conditional Alert

Concept Covered	**Bash: Piping, df -h, awk, Numeric comparison, Conditional logic (if). Python: subprocess module, Regular Expressions (re) or String splitting, Float conversion, Conditional logic.**
Description	Write a script that checks the disk usage for the root partition (/). The script must extract the percentage usage and compare it to a user-defined threshold (e.g., 85%). If the usage is above the threshold, print a warning message; otherwise, print a success message.
Bash Implementation	Use df -h / and pipe the output to awk to extract the usage percentage column and clean the percentage sign (%). Store the result in a variable and use a numerical if test (-gt) for comparison.
Python Implementation	Use subprocess to run df -h /. Capture the

	output, use string splitting or a regular expression to find and extract the percentage number, convert it to an integer/float, and use a conditional check.

14. Variable Scope Test (Advanced)

Concept Covered	**Bash: Functions, local keyword, Subshells (()), Exported variables (export). Python: Functions, Global vs. Local scope, global keyword.**
Description	Define a global variable X=10. Create three functions: 1) func_local: Sets X=20 using the local keyword. 2) func_global: Sets X=30 without the local keyword. 3) func_subshell: Executes a new script/command in a subshell that tries to read and modify X (requires export). After calling each function, print the value of X outside the function to demonstrate the variable's scope.
Bash Implementation	Explicitly define the three functions as described, paying attention to the local keyword and the use of the export command for the subshell test.
Python Implementation	Define the three functions. Use the global keyword in func_global to explicitly modify the module-level variable. func_local will automatically use local scope. For the "subshell" equivalent, use subprocess to run a Python script that imports the variable via environment/command-line to mimic the scope separation.