Homework 1:
Maria Angel Palacios Sarmiento

Explanation 1:

In this script, I started by greeting the user and asking for their name to make the experience more personal. Then, I prompted the user to start the game, and if they chose "y," I explained the rules of Rock, Paper, Scissors. After ensuring they understood the rules, I asked them to pick an option and used a while loop to validate their input, ensuring they typed it in lowercase. This prevents any input errors and maintains consistency in the game.

For the main game logic, I used an array to store the three options (rock, scissors, paper). This made it easier to randomly select one of the choices for the computer using the RANDOM function, as I didn't have to list each option separately. Using an array also made the code more flexible, so I could easily expand the game in the future by adding more choices without changing much of the structure. The conditional statements then compared the user's choice with the computer's to determine the outcome, ensuring the game runs smoothly.

Explanation 2:

This script checks if the argument given is a file or a directory and then does different things based on that. First, it checks if there's an argument and figures out if it's a directory or file. If it's a directory, it checks if it has the right read and execute permissions. If the permissions are fine, it counts how many subdirectories and files there are, lists them, and shows their sizes using commands like ls and du. For files, it checks the permissions and then uses wc to count the words, characters, and lines in the file. Depending on the character count, the file is labeled as **"Text Heavy," "Moderately Sized,"** or **"Light file."**

The script also handles errors in case the argument is invalid or if there are missing permissions, giving clear warning messages. This ensures everything runs smoothly and the user gets helpful feedback. The structure is flexible, so it can be easily modified in the future for other tasks. I used a loop to list the files because when I tried using du by itself, it included the directory itself in the output, which wasn't what I wanted. After debugging, I figured I had to use cut to grab just the right column and grep to remove the directory. This way, it made sure only the file sizes showed up, and the loop let me apply the same process to each file.

Explanation 3:

This script is set up to handle file backup tasks. First, it checks if at least three command-line arguments (file names) are provided. If not, it prints a message and exits. If there are enough arguments, it stores them in an array. Then, the script goes through each file and checks if it exists. If a file doesn't exist, it gives a warning and moves on to the next file, with the option to create the missing file.

Next, the script creates a backup directory with a timestamp and starts the backup process. It loops through the files again, creating any missing files and copying them into the backup directory with a timestamp added to the file name. It also logs each backup operation, showing which file was backed up and the new filename. Finally, the script tells the user the backup is complete and shows the location of the backup and the log file.

I used "$@" because it makes it easy to pass all the command-line arguments into the function. "$@" lets the script handle each argument individually, even if the file names have spaces. This is helpful because I don't have to manually specify each file name. It makes the code cleaner and more flexible, so I can handle however many files are passed into the script.

Explanation 4:

This Bash script allows users to navigate a directory, view its contents, and interact with files or subdirectories. It first prompts the user for a directory name, checks if it exists, and either navigates into it or offers the option to create it. Once inside, it lists all files and subdirectories, letting the user choose one to interact with. If the selection is a file, the script displays the first 10 lines and asks if the user wants to see more. If the user agrees, it continues displaying the next 10 lines at a time until the user decides to stop or the end of the file is reached. This is handled using the head command for the initial display and a line_num variable to track progress, ensuring smooth scrolling without errors.

If the selected item is a subdirectory, the script navigates into it and performs a recursive search for files modified in the last 24 hours. It presents the results to the user and asks them to select a file, which then follows the same step-by-step viewing process as before. Once the user finishes interacting with a file or subdirectory, the script returns to the main menu, allowing them to select another item or exit. This loop ensures continuous navigation and interaction, making the script a flexible tool for browsing and inspecting files efficiently.