1. Breakdown of How the Script Handles Arguments and Options

The script uses a combination of getopts and positional arguments to handle user input:

- it checks if the first argument is --help. If so, it immediately prints a usage guide and exits.
- It then uses getopts to parse short options –n
    - If -n is detected, it sets nflag=true.
    - If -v is detected, it sets vflag=true.
- After parsing options, the script uses shift to remove them from the argument list, so that the next two arguments are guaranteed to be the pattern and the file.
- It then checks:
    - If exactly two arguments are provided (pattern and filename).
    - If the file exists and is readable.
- Once validation is passed, the script reads the file line-by-line, applies matching logic

2. How Would the Structure Change to Support Regex, -i, -c, or -l Options?

If new regex support will be added like –l, -c or –l, I will use [[$line =~ $pattern]] to support these new updates.

1. for –i, Instead of enabling shopt -s nocasematch, I modify matching behavior based on the presence of -i.
2. For –c, I will add a counter and ++ it every time it matches a line, then print the counter
3. For –l, if a line is found that matches, the script will print the filename and move to the next file.

3. Hardest Part of the Script to Implement

The most challenging part was creating the logic for inverting matches based on multiple conditions

```
if $vflag; then
    if $match; then match=false; else match=true; fi
  fi

  #  If the line should be printed
  if $match; then
```

```
    # If -n is set, print the line number and the line
    # Otherwise, just print the line
    if $nflag; then
      printf "%d:%s\n" "$lineno" "$line"
    else
      printf "%s\n" "$line"
    fi
  fi
```

Because normally, if a line matches the pattern, you print it. But when the –v flag is set, I needed to flip the meaning, also I needed to check if –n set, and print differently based on that.