1. Write a shell script to check whether a number is an Armstrong number (also called a narcissistic number).
Ans-

```
#!/bin/bash

read -p "Enter a number: " num
sum=0
temp=$num
len=${#num}

while [ $temp -gt 0 ]
do
  digit=$((temp % 10))
  # Calculate digit^len using bc
  power=$(echo "$digit ^ $len" | bc)
  sum=$((sum + power))
  temp=$((temp / 10))
done

if [ $sum -eq $num ]; then
  echo "$num is an Armstrong number."
else
  echo "$num is NOT an Armstrong number."
fi
```

---

2. Create a text file name notes.txt and insert the following line.

```
apple orange banana
APPLE GRAPE
red apple pie
pineapple juice
banana split
```

1. Display lines containing either "pie" or "juice".
Ans - grep -E "pie|juice" notes.txt

2. Find lines that end with the letter "e".
Ans - grep "e$" notes.txt

3. Extract the first word in ALL-CAPS from the file.
Ans - grep -o "[A-Z]\+" notes.txt | head -n1

4. Show the last line containing a word starting with ’b’
Ans - grep "\bb" notes.txt | tail -n1

5. Count how many times the letter ’a’ appears in the first 3 lines.
Ans - head -n3 notes.txt | grep -o "a" | wc -l

Q1) Create two branches, branch-A and branch-B, from the main branch.
On branch-A, edit file1.txt and add the line Feature added by branch-A. Commit the changes.

On branch-B, edit the same line in file1.txt to Feature added by branch-B. Commit the changes.
Merge branch-B into branch-A and resolve the conflict manually if any.

Ans - git checkout main
git checkout -b branch-A
Create and Edit file1.txt and add:Feature added by branch-A
git add file1.txt
git commit -m "Add feature line from branch-A"
git checkout main
git checkout -b branch-B
Edit the same line in file1.txt to: Feature added by branch-B
git add file1.txt
git commit -m "Add feature line from branch-B"
git checkout branch-A
git merge branch-B - This will result in a merge conflict in file1.txt, since both branches edited the same line.
Manually resolve the conflict
git add file1.txt
git commit -m "Merge branch-B into branch-A and resolve conflict in file1.txt"

Q2) Edit the file undo_changes.txt by adding some text and stage it.
Undo the staged changes and restore the file to its previous state.
Now edit the file again and commit the changes.
Then undo the last commit while retaining the changes in the working directory.

Ans - echo "Some new content" >> undo_changes.txt
git add undo_changes.txt

git restore --staged undo_changes.txt
git restore undo_changes.txt

echo "Final committed content" >> undo_changes.txt
git add undo_changes.txt
git commit -m "Add final committed content to undo_changes.txt"

git reset --mixed HEAD~1

To verify:
git status - O/P will be - Changes not staged for commit: (use "git add <file>..." to update what will be committed)

Q3) Edit two files: fileA.txt and fileB.txt.
Do not stage the changes.
Use Git to stash the changes temporarily.
Edit another file, fileC.txt, stage, and commit it.
Apply the stashed changes back and verify the files.

Ans - echo "Change in fileA" >> fileA.txt
echo "Change in fileB" >> fileB.txt

git stash push -m "Temporary edits to fileA and fileB" -u

```
echo "Update in fileC" >> fileC.txt
git add fileC.txt
git commit -m "Update fileC.txt"

git stash apply
```

To verify:

git status - O/P will be - You should see fileA.txt and fileB.txt listed under "Changes not staged for commit", meaning their stashed content is restored.