Q1: LCM

```
lcm=1
for num in $(rev $1); do
    a=$lcm
    b=$num
    while [ $b -ne 0 ]
    do
        t=$b
        b=$(($a % $b))
        a=$t
    done
    lcm=$(($lcm*$(($num/$a))))
done
echo $lcm
```

soln:
This is a shell script in the bash scripting language that calculates the least common multiple (LCM) of a list of numbers.

The script takes one argument, $1, which is a list of numbers, separated by spaces, passed to the script. The rev command reverses the order of the numbers in the list.

The script uses a loop to iterate over each number in the list, stored in the variable num. Inside the loop, the variable lcm is initialized to 1 and is updated in each iteration with the LCM of the previous iteration's result and the current number num.

The LCM calculation is performed using a Euclidean algorithm to find the greatest common divisor (GCD) of lcm and num. The GCD is then used to calculate the LCM by multiplying lcm and num and dividing the result by their GCD.

Finally, the script outputs the final value of lcm, which is the LCM of all numbers in the list.

Q2:

```
#!/bin/bash
while read username;
do
    [[ $username =~ ^.{5,20}$ ]] && [[ $username =~ ^[a-zA-Z][a-zA-Z0-9]*[0-9][a-zA-Z0-9]*$ ]] && !
grep -iqFf fruits.txt <<< "$username" && echo "YES" >> validation_results.txt || echo "NO" >>
validation_results.txt
done < $1
```

Sol:
This is a shell script in the bash scripting language that validates usernames from a file and outputs the results to another file.

The script reads each line of the input file $1 and stores the line in the variable username. Then, it performs a regular expression match to check if username satisfies two conditions:

It has a length of between 5 and 20 characters. This is checked with the expression ^.{5,20}$.

It starts with a letter (upper or lower case), followed by zero or more letters or numbers, followed by a number, followed by zero or more letters or numbers. This is checked with the expression ^[a-zA-Z][a-zA-Z0-9]*[0-9][a-zA-Z0-9]*$.

If username satisfies the two conditions, the script then uses the grep command with the -iqFf options to check if username appears in a file named fruits.txt in a case-insensitive fashion. If it does not, the script outputs "YES" to a file named validation_results.txt.

If username does not satisfy one or both of the conditions or appears in the fruits.txt file, the script outputs "NO" to the validation_results.txt file.

Q3:

```
json_dir=$1
csv_dir=$2
attributes=("$@")

attributes=("${attributes[@]:2}")

for file in "$json_dir"/*
do
  base_file_name=$(basename "$file" .jsonl)
  csv_file_path="$csv_dir/$base_file_name.csv"
  header=""
for attribute in "${attributes[@]}"
do
  header="$header,$attribute"
done
echo "${header:1}" > "$csv_file_path"
  while read -r line
  do
    csv_row=""
    for attribute in "${attributes[@]}"
    do
      value=$(echo "$line" | jq -r ".$attribute")
      if [[ $value == *","* ]]
      then
        value="\"$value\""
      fi
      csv_row="$csv_row,$value"
    done
    echo "${csv_row:1}" >> "$csv_file_path"
  done < "$file"
done
```

Sol:
This is a bash shell script that converts JSONL (newline-delimited JSON) files in a specified directory (json_dir) to CSV (Comma-Separated Values) files in a specified directory (csv_dir). The script takes a list of attributes as command line arguments, and converts each JSON object in the JSONL file to a row in the CSV file, using the specified attributes as column names in the CSV file.

The script performs the following actions:

Sets the first command line argument $1 to json_dir and the second command line argument $2 to csv_dir.
Extracts the list of attributes from the command line arguments by slicing the attributes array from the third argument to the end.
For each JSONL file in the json_dir directory:
a. Extracts the base file name of the JSONL file without the '.jsonl' extension and constructs the path of the corresponding CSV file in the csv_dir directory.
b. Creates a header row for the CSV file using the list of attributes.
c. Reads each line in the JSONL file, extracts the values of the specified attributes, and writes a row to the CSV file.
d. If the value of an attribute contains a comma, it is enclosed in double quotes.

Q4

```
#!/bin/bash
while read line; do
    if echo $line | grep -q $2; then
        echo "$line" | sed 's/.*/\U&/' | sed -e 's/\([A-Z][^a-zA-Z]*\)\([A-Z]\)/\1\l\2/g'
    else
        echo $line
    fi
done < $1
```

Sol:
This script takes two arguments: a file name ($1) and a string ($2). It reads each line from the file and checks if the line contains the string. If it does, the line is transformed to have the first character of each word capitalized. If the line does not contain the string, it is printed as-is. The transformed or original lines are then printed to the standard output.

Q5:

```
folder=$1

if [ -z "$folder" ]; then
    echo "Usage: $0 <folder>"
    exit 1
fi

if [ ! -d "$folder" ]; then
    echo "Error: $folder is not a directory"
    exit 1
fi

for file in $(find $folder -name "*.py"); do
    echo $file
   flag=0
    n=0
    output=""
    while read -r word; do

      if [[ $word == *'"""'* ]]; then
        if [ $flag -eq 0 ]; then
          flag=1
          output="$output $word"
          temp=0
            while IFS= read -r line; do
            temp=$((temp+1))
        if [ $temp -gt $n ]; then


        n=$((n+1))
        if [[ $line == *"$word"* ]]; then
              echo "line $n:"
              break
          fi
        fi
        done < "$file"
        else
          flag=0
          output="$output $word"
          echo "$output"
          output=""
        fi
      elif [ $flag -eq 1 ]; then
        output="$output $word"
      else
        if [ -n "$output" ]; then
          echo "$output"
          output=""
        fi
      fi
    done < <(tr -s '[[:space:]]' '\n' < $file)
    if [ -n "$output" ]; then
      echo "$output "
    fi
awk '{if(/^[^'\''"]*#/) {gsub(/^[^#]+#/, "#", $0);print "Line " NR ": " $0;} else if(/#[^'\''"]*$/) {gsub(/
^.*#/, "#", $0);print "Line " NR ": " $0;} else if(/^.*#[^'\''"]*$/) {gsub(/^.*#/, "#", $0);print "Line " NR
":  "  $0;}}' $file
done
```

Sol:
This code block searches for all Python files in a specified folder, and then prints the contents of each file with the line number preceding each line that starts with a comment symbol "#".

The first if statement checks if the folder argument is present. If it is not present, the code prints an error message and exits with status code 1.
The second if statement checks if the folder argument is a directory. If it is not a directory, the code prints an error message and exits with status code 1.
The for loop uses the find command to locate all files with a ".py" extension in the specified folder.
The code inside the for loop performs the following actions for each file:
Reads each line of the file, one at a time, stored in the variable word.
The if statement checks if the line contains the string """. If it does, the code sets the flag variable to 1, indicating that the line is the start of a multiline string.
The inner while loop reads each line of the file again, one at a time. The loop continues until the end of the multiline string is found.
If the line number of the current line is greater than the line number of the start of the multiline string, the code checks if the line contains the end of the multiline string. If it does, the code prints the line number and breaks out of the inner loop.
The elif statement checks if the flag variable is equal to 1, indicating that the line is part of the multiline string. The line is added to the output variable.
The final else statement handles all other lines that are not part of a multiline string. If the output variable is not empty, the code prints the contents of the output variable and resets it to an empty string.
After all lines of the file have been processed, the code checks if the output variable is not empty. If it is not empty, the code prints the contents of the output variable.
Finally, the awk command is used to print the line number and contents of each line that starts with a comment symbol “#".

Q6::

```
#!/bin/bash
declare -a minPrimeFactor

Sieve_of_Eratosthenes() {
   for((i=2; i<=1000000; i++))
   do
      minPrimeFactor[i]=$i
   done
   for((i=4; i<=1000000; i=i+2))
   do
      minPrimeFactor[i]=2
   done
   for((i=3; i*i<=1000000; i=i+2))
   do
      if [[ ${minPrimeFactor[i]} -eq $i ]]; then
         for((j=i*i; j<=1000000; j+=i))
         do
         if [ ${minPrimeFactor[j]} -eq $j ]; then
            minPrimeFactor[j]=$i
         fi
         done
      fi
   done
}

Sieve_of_Eratosthenes
while read -r x
do
   declare -a factors
   while [[ $x -ne 1 ]]
   do
      factors[minPrimeFactor[x]]=1
      x=$((x/minPrimeFactor[x]))
   done
   for i in "${!factors[@]}"; do
      echo -n "$i " >> output.txt
   done
   echo >> output.txt
   factors=()
done < input.txt
```

SOL::
This is a bash script that performs a prime factorization of a set of numbers. The script starts by generating an array of minimum prime factors using the Sieve of Eratosthenes algorithm. Then it reads numbers from a file named "input.txt", performs prime factorization on each number, and writes the results to a file named "output.txt". The prime factorization of a number is done by dividing the number by its minimum prime factor repeatedly until it can no longer be divided, and each minimum prime factor encountered is recorded as a factor of the original number.

Q7::

```
#!/bin/bash
mkdir -p "$2"
for letter in {a..z}; do
    grep -rih "^[$letter]" $1/*.txt | sort > "$2/$letter".txt
done
```

Sol::

This script creates a new directory specified in the second argument and populates it with 26 separate files, one for each letter of the alphabet, from a to z.

It does this by using a for loop to iterate over each letter in the alphabet. For each letter, it runs the grep command to search for lines in all .txt files in the first argument directory that start with the current letter. The -r flag tells grep to search the input directories recursively, the -i flag tells grep to ignore the case of the letters being searched for, and the -h flag

Q9::

```
#!/bin/bash
awk '{c[$2]++} END {for (i in c) {print i " " c[i]} print ""}' $1 | sort -k2nr -k1
awk '{c[$1]++} END {for (i in c) {if(c[i]>=2) {print i} if(c[i]==1) {n++}} print n}' $1
```

This is a bash script that uses awk to count the occurrences of the second field in the input file and print the result in a sorted manner, with the number of occurrences in descending order, and then the second awk command counts the number of unique first fields in the input file. The result is the number of unique first fields.