CDAC MUMBAI

Concepts of Operating System
Assignment 2

Part A

What will the following commands do?

1. `echo "Hello, World!"`: Prints "Hello, World!" to the terminal.

2. `name="Productive"`: Assigns the string "Productive" to a variable named `name`.

3. `touch file.txt`: Creates an empty file named `file.txt` if it doesn't already exist.

4. `ls -a`: Lists all files and directories in the current directory, including hidden files (those starting with a dot).

5. `rm file.txt`: Deletes the file named `file.txt`.

6. `cp file1.txt file2.txt`: Copies the contents of `file1.txt` to `file2.txt`. If `file2.txt` doesn't exist, it will be created.

7. `mv file.txt /path/to/directory/`: Moves `file.txt` to the specified directory.

8. `chmod 755 script.sh`: Changes the permissions of `script.sh` to `rwxr-xr-x`, which means read, write, and execute permissions for the owner, and read and execute permissions for the group and others.

9. `grep "pattern" file.txt`: Searches for occurrences of the string "pattern" in `file.txt` and prints the matching lines.

10. `kill PID`: Terminates the process with the specified Process ID (PID).

11. `mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt`: Creates a directory named `mydir`, changes to that directory, creates an empty file named `file.txt`, writes "Hello, World!" to `file.txt`, and then prints the contents of `file.txt` to the terminal.

12. `ls -l | grep ".txt"`: Lists all files in long format and filters the output to show only files with `.txt` in their

name.

13. `cat file1.txt file2.txt | sort | uniq`: Concatenates `file1.txt` and `file2.txt`, sorts the combined content, and removes duplicate lines.

14. `ls -l | grep "^d"`: Lists all files in long format and filters the output to show only directories.

15. `grep -r "pattern" /path/to/directory/`: Recursively searches for the string "pattern" in all files within the specified directory and its subdirectories.

16. `cat file1.txt file2.txt | sort | uniq -d`: Concatenates `file1.txt` and `file2.txt`, sorts the combined content, and prints only the duplicate lines.

17. `chmod 644 file.txt`: Changes the permissions of `file.txt` to `rw-r--r--`, which means read and write permissions for the owner, and read-only permissions for the group and others.

18. `cp -r source_directory destination_directory`: Copies the contents of `source_directory` to `destination_directory` recursively, including all files and subdirectories.

19. `find /path/to/search -name "*.txt"`: Searches for all files with a `.txt` extension within the specified directory and its subdirectories.

20. `chmod u+x file.txt`: Adds execute permissions to the owner of `file.txt`.

21. `echo $PATH`: Prints the value of the `PATH` environment variable, which lists the directories the shell searches for executable files.

Part B

Identify True or False:

1. `ls` is used to list files and directories in a directory. **True**

2. `mv` is used to move files and directories. **True**

3. `cd` is used to copy files and directories. **False** (`cd` is used to change the current directory.)

4. `pwd` stands for "print working directory" and displays the current directory. **True**

5. `grep` is used to search for patterns in files. **True**

6. `chmod 755 file.txt` gives read, write, and execute permissions to the owner, and read and execute permissions to the group and others. **True**

7. `mkdir -p directory1/directory2` creates nested directories, creating `directory2` inside `directory1` if `directory1` does not exist. **True**

8. `rm -rf file.txt` deletes a file forcefully without confirmation. **True**

Identify the Incorrect Commands:

1. `chmodx` is used to change file permissions. **Incorrect** (The correct command is `chmod`.)

2. `cpy` is used to copy files and directories. **Incorrect** (The correct command is `cp`.)

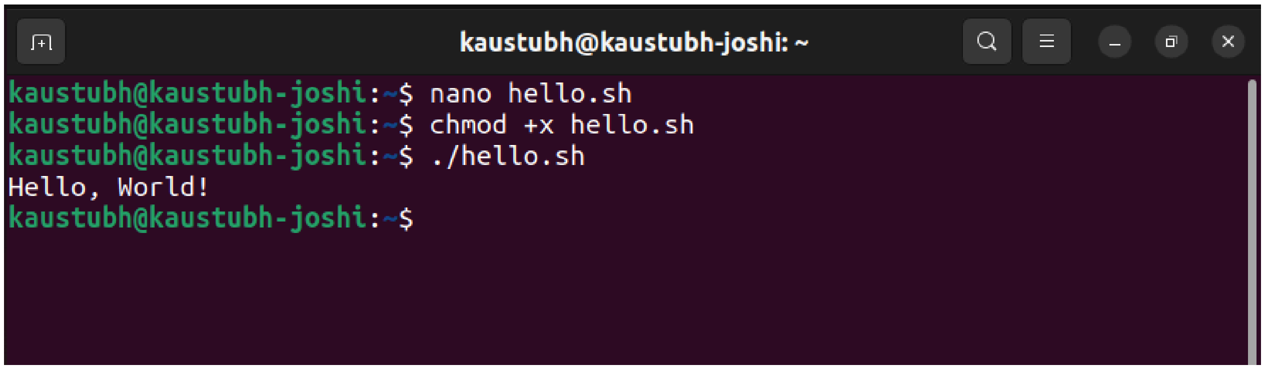
3. `mkfile` is used to create a new file. **Incorrect** (There is no standard command `mkfile`. The correct way to create a new file is using `touch`.)

4. `catx` is used to concatenate files. **Incorrect** (The correct command is `cat`.)

5. `rn` is used to rename files. **Incorrect** (The correct command is `mv`.)

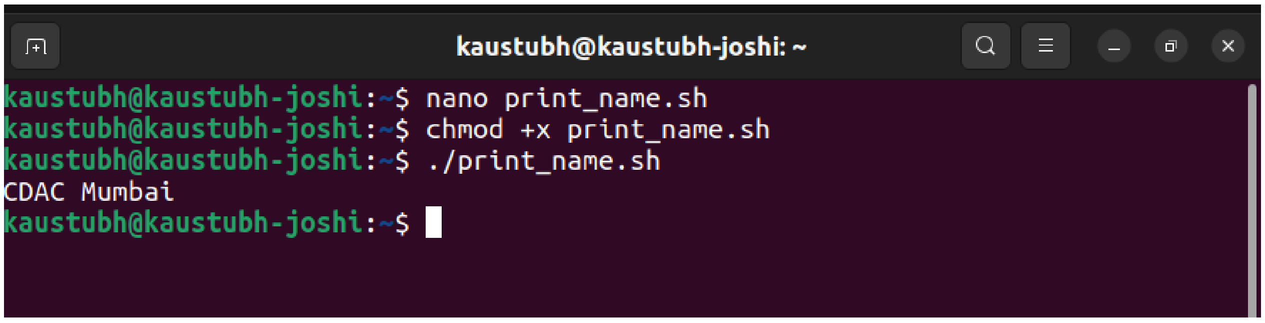
Part C

Question 1: Write a shell script that prints "Hello, World!" to the terminal.



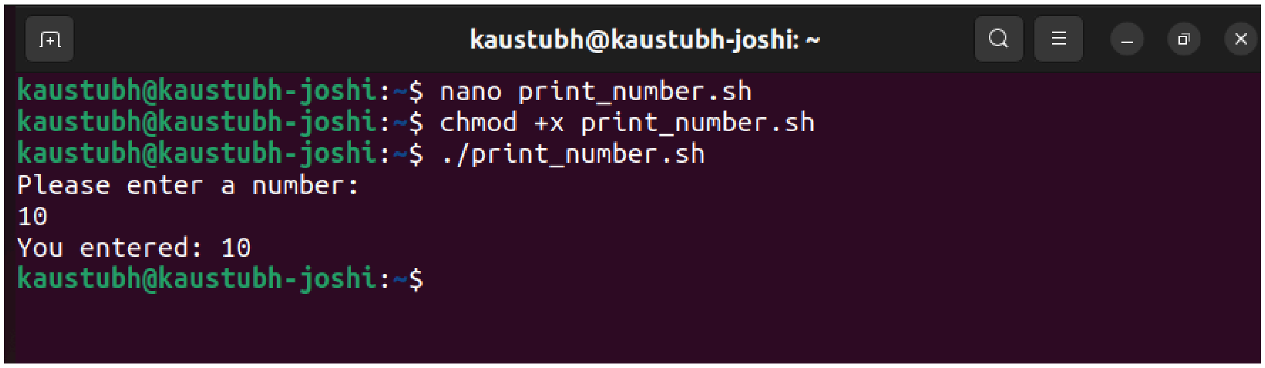
```
kaustubh@kaustubh-joshi: ~
kaustubh@kaustubh-joshi:~$ nano hello.sh
kaustubh@kaustubh-joshi:~$ chmod +x hello.sh
kaustubh@kaustubh-joshi:~$ ./hello.sh
Hello, World!
kaustubh@kaustubh-joshi:~$
```

Question 2: Declare a variable named "name" and assign the value "CDAC Mumbai" to it. Print the value of the variable.



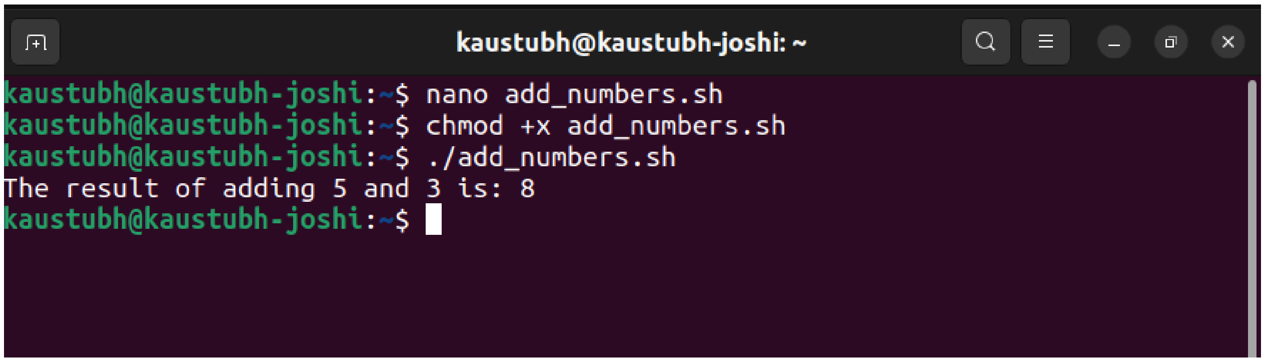
```
kaustubh@kaustubh-joshi: ~
kaustubh@kaustubh-joshi:~$ nano print_name.sh
kaustubh@kaustubh-joshi:~$ chmod +x print_name.sh
kaustubh@kaustubh-joshi:~$ ./print_name.sh
CDAC Mumbai
kaustubh@kaustubh-joshi:~$
```

Question 3: Write a shell script that takes a number as input from the user and prints it.



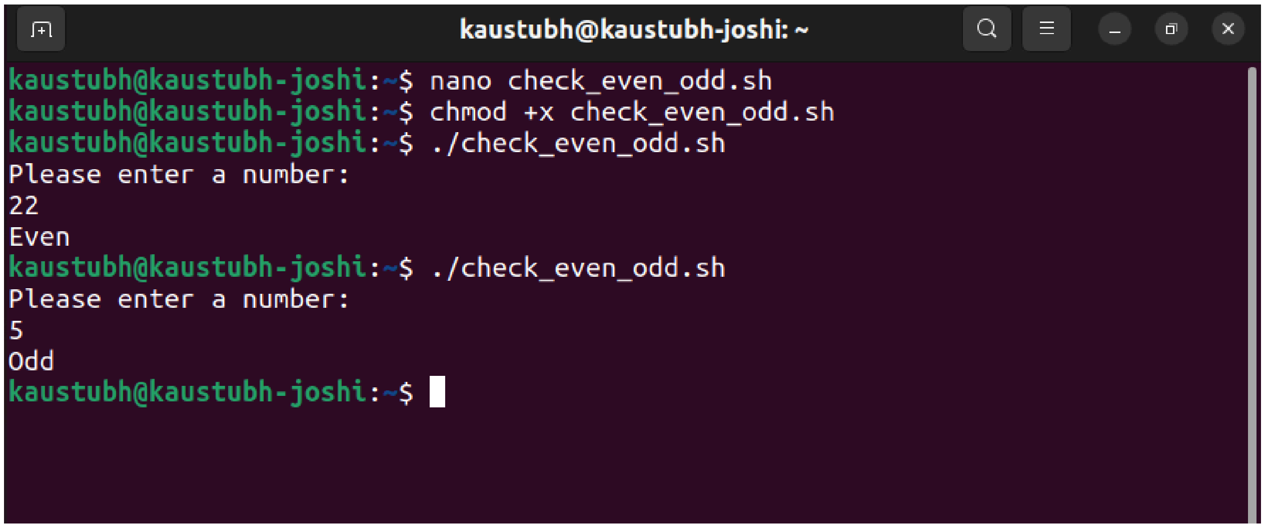
```
kaustubh@kaustubh-joshi:~$ nano print_number.sh
kaustubh@kaustubh-joshi:~$ chmod +x print_number.sh
kaustubh@kaustubh-joshi:~$ ./print_number.sh
Please enter a number:
10
You entered: 10
kaustubh@kaustubh-joshi:~$
```

Question 4: Write a shell script that performs addition of two numbers (e.g., 5 and 3) and prints the result.



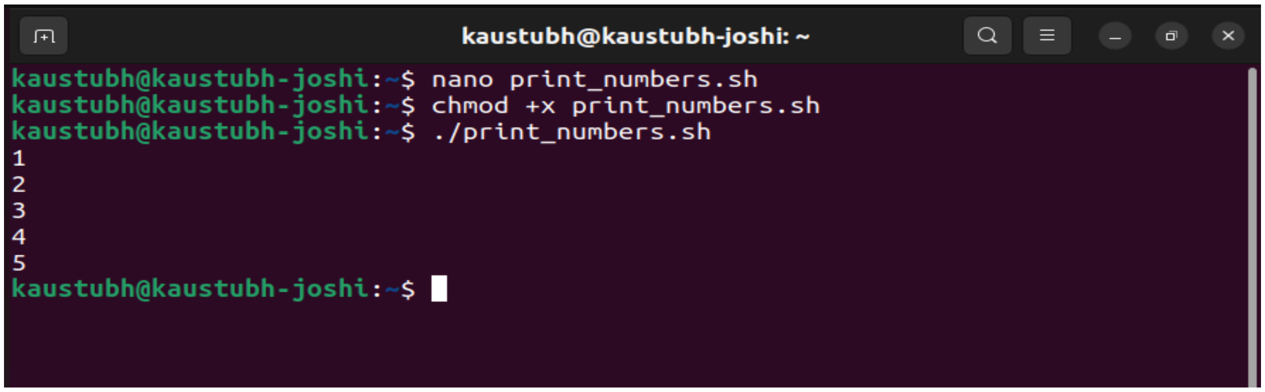
```
kaustubh@kaustubh-joshi:~$ nano add_numbers.sh
kaustubh@kaustubh-joshi:~$ chmod +x add_numbers.sh
kaustubh@kaustubh-joshi:~$ ./add_numbers.sh
The result of adding 5 and 3 is: 8
kaustubh@kaustubh-joshi:~$
```

Question 5: Write a shell script that takes a number as input and prints "Even" if it is even, otherwise prints "Odd".



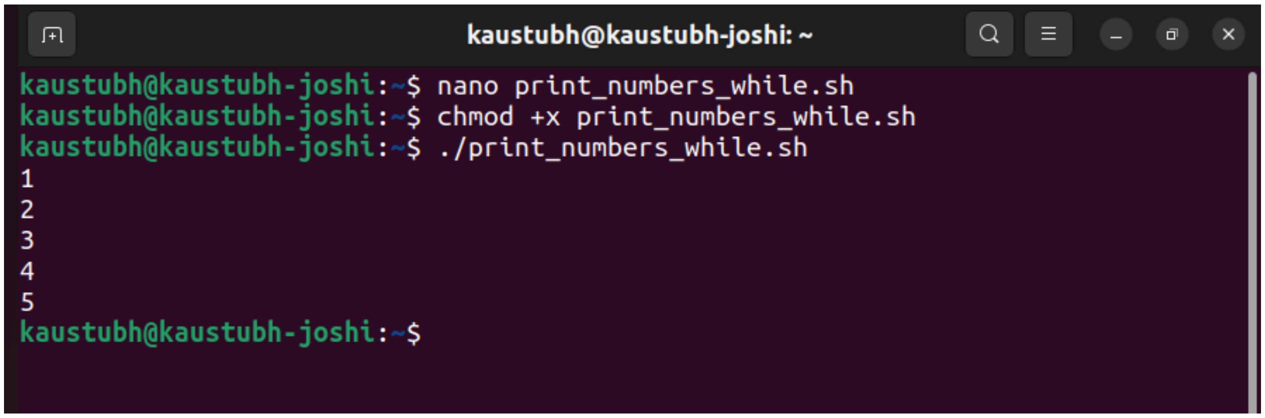
```
kaustubh@kaustubh-joshi:~$ nano check_even_odd.sh
kaustubh@kaustubh-joshi:~$ chmod +x check_even_odd.sh
kaustubh@kaustubh-joshi:~$ ./check_even_odd.sh
Please enter a number:
22
Even
kaustubh@kaustubh-joshi:~$ ./check_even_odd.sh
Please enter a number:
5
Odd
kaustubh@kaustubh-joshi:~$
```

Question 6: Write a shell script that uses a for loop to print numbers from 1 to 5.



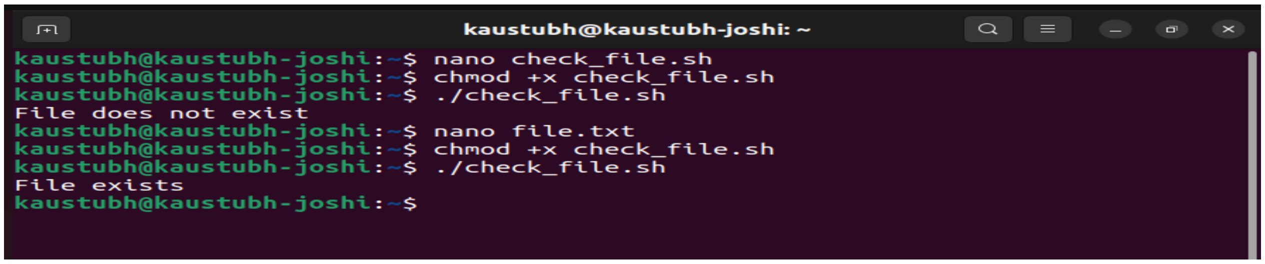
```
kaustubh@kaustubh-joshi: ~
kaustubh@kaustubh-joshi:~$ nano print_numbers.sh
kaustubh@kaustubh-joshi:~$ chmod +x print_numbers.sh
kaustubh@kaustubh-joshi:~$ ./print_numbers.sh
1
2
3
4
5
kaustubh@kaustubh-joshi:~$
```

Question 7: Write a shell script that uses a while loop to print numbers from 1 to 5.



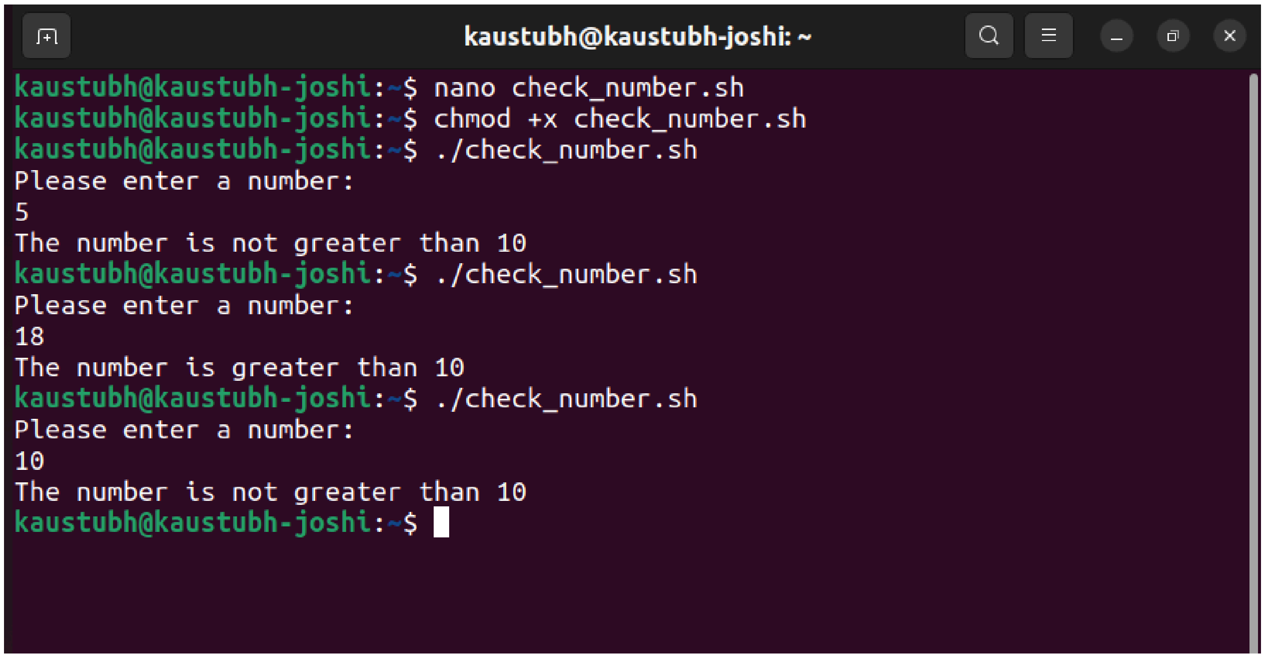
```
kaustubh@kaustubh-joshi: ~
kaustubh@kaustubh-joshi:~$ nano print_numbers_while.sh
kaustubh@kaustubh-joshi:~$ chmod +x print_numbers_while.sh
kaustubh@kaustubh-joshi:~$ ./print_numbers_while.sh
1
2
3
4
5
kaustubh@kaustubh-joshi:~$
```

Question 8: Write a shell script that checks if a file named "file.txt" exists in the current directory. If it does, print "File exists", otherwise, print "File does not exist".



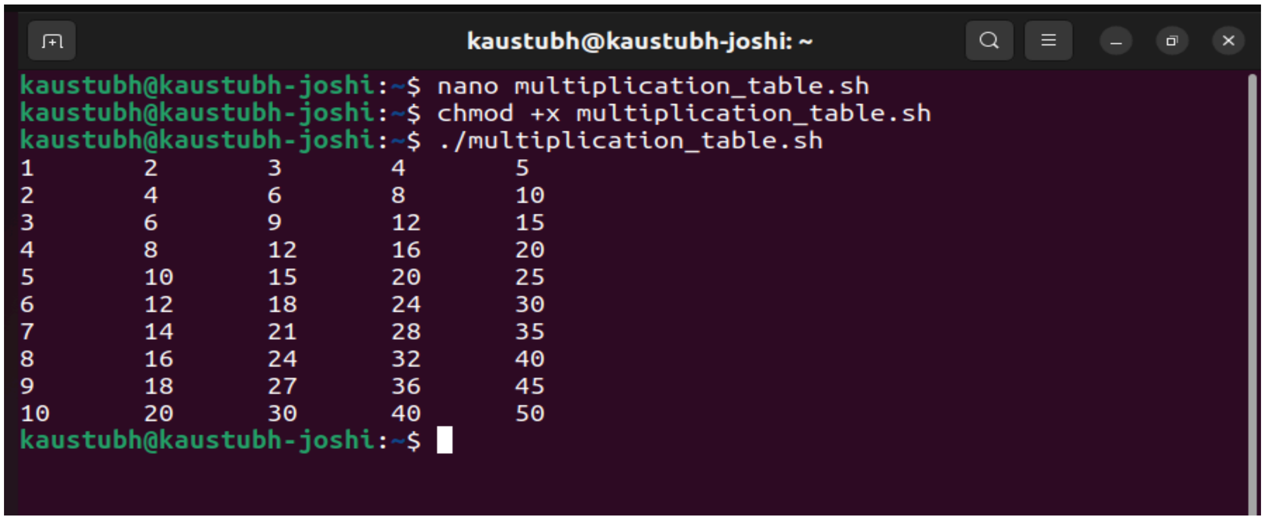
```
kaustubh@kaustubh-joshi: ~
kaustubh@kaustubh-joshi:~$ nano check_file.sh
kaustubh@kaustubh-joshi:~$ chmod +x check_file.sh
kaustubh@kaustubh-joshi:~$ ./check_file.sh
File does not exist
kaustubh@kaustubh-joshi:~$ nano file.txt
kaustubh@kaustubh-joshi:~$ chmod +x check_file.sh
kaustubh@kaustubh-joshi:~$ ./check_file.sh
File exists
kaustubh@kaustubh-joshi:~$
```

Question 9: Write a shell script that uses the if statement to check if a number is greater than 10 and prints a message accordingly.



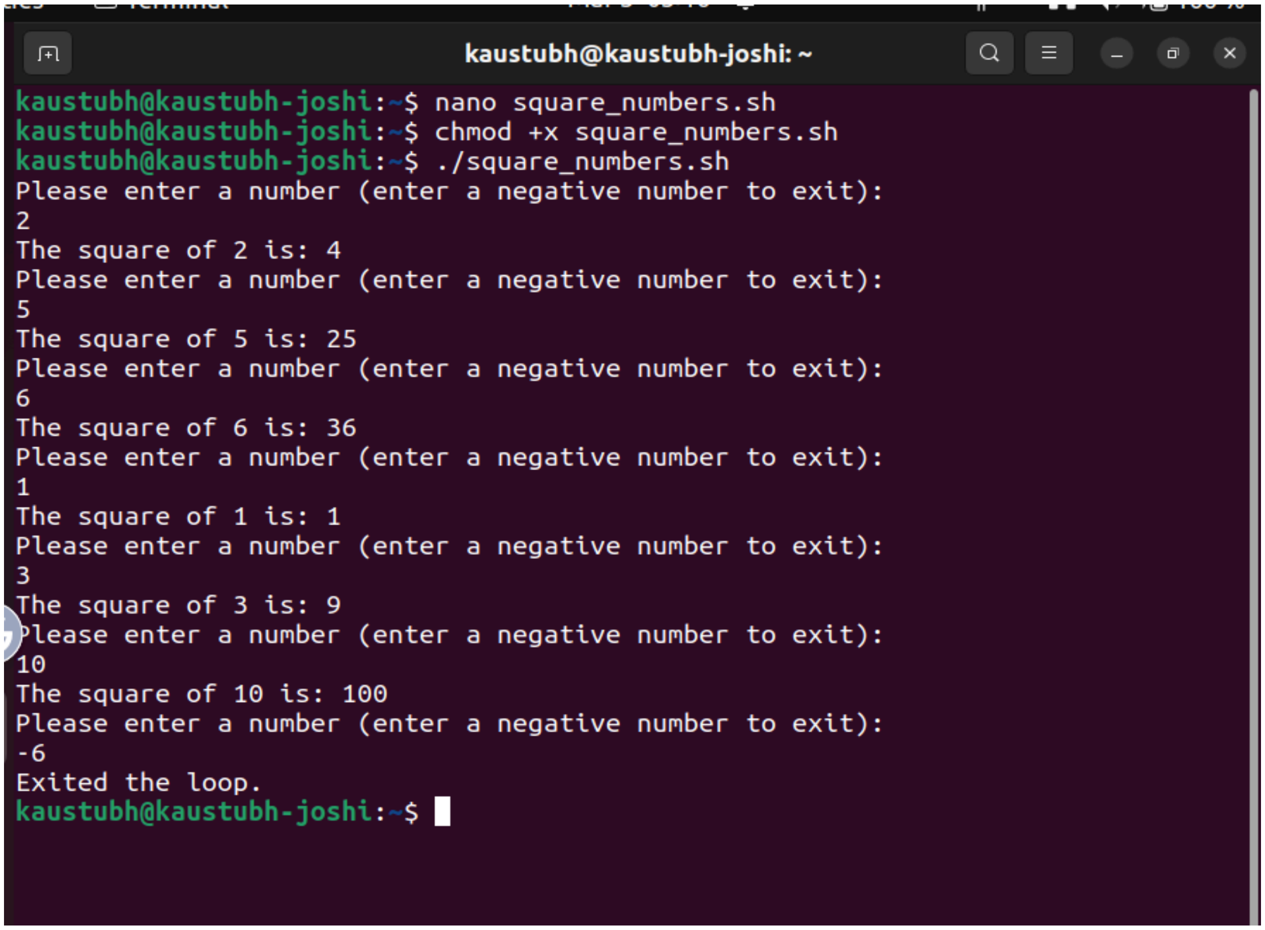
```
kaustubh@kaustubh-joshi:~$ nano check_number.sh
kaustubh@kaustubh-joshi:~$ chmod +x check_number.sh
kaustubh@kaustubh-joshi:~$ ./check_number.sh
Please enter a number:
5
The number is not greater than 10
kaustubh@kaustubh-joshi:~$ ./check_number.sh
Please enter a number:
18
The number is greater than 10
kaustubh@kaustubh-joshi:~$ ./check_number.sh
Please enter a number:
10
The number is not greater than 10
kaustubh@kaustubh-joshi:~$
```

Question 10: Write a shell script that uses nested for loops to print a multiplication table for numbers from 1 to 5. The output should be formatted nicely, with each row representing a number and each column representing the multiplication result for that number.



```
kaustubh@kaustubh-joshi:~$ nano multiplication_table.sh
kaustubh@kaustubh-joshi:~$ chmod +x multiplication_table.sh
kaustubh@kaustubh-joshi:~$ ./multiplication_table.sh
1       2       3       4       5
2       4       6       8       10
3       6       9       12      15
4       8       12      16      20
5       10      15      20      25
6       12      18      24      30
7       14      21      28      35
8       16      24      32      40
9       18      27      36      45
10      20      30      40      50
kaustubh@kaustubh-joshi:~$
```

Question 11: Write a shell script that uses a while loop to read numbers from the user until the user enters a negative number. For each positive number entered, print its square. Use the break statement to exit the loop when a negative number is entered.



```
kaustubh@kaustubh-joshi:~$ nano square_numbers.sh
kaustubh@kaustubh-joshi:~$ chmod +x square_numbers.sh
kaustubh@kaustubh-joshi:~$ ./square_numbers.sh
Please enter a number (enter a negative number to exit):
2
The square of 2 is: 4
Please enter a number (enter a negative number to exit):
5
The square of 5 is: 25
Please enter a number (enter a negative number to exit):
6
The square of 6 is: 36
Please enter a number (enter a negative number to exit):
1
The square of 1 is: 1
Please enter a number (enter a negative number to exit):
3
The square of 3 is: 9
Please enter a number (enter a negative number to exit):
10
The square of 10 is: 100
Please enter a number (enter a negative number to exit):
-6
Exited the loop.
kaustubh@kaustubh-joshi:~$
```

Part D

Common Interview Questions (Must know)

1. What is an operating system, and what are its primary functions?
2. Explain the difference between process and thread.
3. What is virtual memory, and how does it work?
4. Describe the difference between multiprogramming, multitasking, and multiprocessing. 5. What is a file system, and what are its components?
6. What is a deadlock, and how can it be prevented?
7. Explain the difference between a kernel and a shell.
8. What is CPU scheduling, and why is it important?
9. How does a system call work?
10. What is the purpose of device drivers in an operating system?
11. Explain the role of the page table in virtual memory management.
12. What is thrashing, and how can it be avoided?
13. Describe the concept of a semaphore and its use in synchronization.
14. How does an operating system handle process synchronization?

15. What is the purpose of an interrupt in operating systems?
16. Explain the concept of a file descriptor.
17. How does a system recover from a system crash?
18. Describe the difference between a monolithic kernel and a microkernel.
19. What is the difference between internal and external fragmentation?
20. How does an operating system manage I/O operations?
21. Explain the difference between preemptive and non-preemptive scheduling. 22. What is round-robin scheduling, and how does it work?
23. Describe the priority scheduling algorithm. How is priority assigned to processes? 24. What is the shortest job next (SJN) scheduling algorithm, and when is it used? 25. Explain the concept of multilevel queue scheduling.
26. What is a process control block (PCB), and what information does it contain? 27. Describe the process state diagram and the transitions between different process states. 28. How does a process communicate with another process in an operating system? 29. What is process synchronization, and why is it important?
30. Explain the concept of a zombie process and how it is created.
31. Describe the difference between internal fragmentation and external fragmentation. 32. What is demand paging, and how does it improve memory management efficiency? 33. Explain the role of the page table in virtual memory management.
34. How does a memory management unit (MMU) work?
35. What is thrashing, and how can it be avoided in virtual memory systems?
36. What is a system call, and how does it facilitate communication between user programs and the operating system?
37. Describe the difference between a monolithic kernel and a microkernel.
38. How does an operating system handle I/O operations?
39. Explain the concept of a race condition and how it can be prevented.
40. Describe the role of device drivers in an operating system.
41. What is a zombie process, and how does it occur? How can a zombie process be prevented? 42. Explain the concept of an orphan process. How does an operating system handle orphan processes?
43. What is the relationship between a parent process and a child process in the context of process management?
44. How does the fork() system call work in creating a new process in Unix-like operating systems? 45. Describe how a parent process can wait for a child process to finish execution. 46. What is the significance of the exit status of a child process in the wait() system call? 47. How can a parent process terminate a child process in Unix-like operating systems? 48. Explain the difference between a process group and a session in Unix-like operating systems. 49. Describe how the exec() family of functions is used to replace the current process image with a new one.
50. What is the purpose of the waitpid() system call in process management? How does it differ from wait()?
51. How does process termination occur in Unix-like operating systems?
52. What is the role of the long-term scheduler in the process scheduling hierarchy? How does it influence the degree of multiprogramming in an operating system?
53. How does the short-term scheduler differ from the long-term and medium-term schedulers in terms of frequency of execution and the scope of its decisions?
54. Describe a scenario where the medium-term scheduler would be invoked and explain how it helps manage system resources more efficiently.

Part E

1. Consider the following processes with arrival times and burst times:

Process	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	6

Calculate the average waiting time using First-Come, First-Served (FCFS) scheduling.
Ans:-

Step-by-Step Calculation:

1. Process P1:
   - Arrival Time: 0
   - Burst Time: 5
   - Start Time: 0 (since it is the first process)
   - Finish Time: 0 + 5 = 5
   - Waiting Time: Start Time - Arrival Time = 0 - 0 = 0

2. Process P2:
   - Arrival Time: 1
   - Burst Time: 3
   - Start Time: 5 (after P1 finishes)
   - Finish Time: 5 + 3 = 8
   - Waiting Time: Start Time - Arrival Time = 5 - 1 = 4

3. Process P3:
   - Arrival Time: 2
   - Burst Time: 6
   - Start Time: 8 (after P2 finishes)
   - Finish Time: 8 + 6 = 14
   - Waiting Time: Start Time - Arrival Time = 8 - 2 = 6

Calculating the Average Waiting Time:

Average Waiting Time = Total Waiting Time/Number of Processes

Total Waiting Time = 0 + 4 + 6 = 10
Number of Processes = 3

Average Waiting Time = 10/3 ≈ 3.33

Therefore, the average waiting time for the given processes using FCFS scheduling is approximately 3.33 units.

2. Consider the following processes with arrival times and burst times:

Process	Arrival Time	Burst Time
P1	0	3
P2	1	5
P3	2	1
P4	3	4

Calculate the average turnaround time using Shortest Job First (SJF) scheduling.
Ans:-

Step-by-Step Calculation:

1. Process P1 (Arrival Time: 0, Burst Time: 3):
   - Start Time: 0 (first process to arrive)
   - Finish Time: 0 + 3 = 3
   - Turnaround Time: Finish Time - Arrival Time = 3 - 0 = 3

2. Process P3 (Arrival Time: 2, Burst Time: 1):
   - Start Time: 3 (after P1 finishes)
   - Finish Time: 3 + 1 = 4
   - Turnaround Time: Finish Time - Arrival Time = 4 - 2 = 2

3. Process P4 (Arrival Time: 3, Burst Time: 4):
   - Start Time: 4 (after P3 finishes)
   - Finish Time: 4 + 4 = 8
   - Turnaround Time: Finish Time - Arrival Time = 8 - 3 = 5

4. Process P2 (Arrival Time: 1, Burst Time: 5):
   - Start Time: 8 (after P4 finishes)
   - Finish Time: 8 + 5 = 13
   - Turnaround Time: Finish Time - Arrival Time = 13 - 1 = 12

Calculating the Average Turnaround Time:
Average Turnaround Time = Total Turnaround Time/Number of Processes
Total Turnaround Time = 3 + 2 + 5 + 12 = 22
Number of Processes = 4
Average Turnaround Time = 22/4 = 5.5

Therefore, the average turnaround time for the given processes using Shortest Job First (SJF) scheduling is 5.5 units.

3. Consider the following processes with arrival times, burst times, and priorities (lower number indicates higher priority):

Process	Arrival Time	Burst Time	Priority
P1	0	6	3
P2	1	4	1
P3	2	7	4
P4	3	2	2

Calculate the average waiting time using Priority Scheduling.
Ans:-

Process	Arrival Time	Burst Time	Priority
P1	0	6	3
P2	1	4	1
P3	2	7	4
P4	3	2	2

Step-by-Step Calculation:

1. Process P1 starts first:
   - Arrival Time: 0
   - Burst Time: 6
   - Priority: 3
   - Start Time: 0
   - Finish Time: 0 + 6 = 6
   - Waiting Time: Start Time - Arrival Time = 0 - 0 = 0

2. Process P2 (highest priority) starts after P1 finishes:
   - Arrival Time: 1
   - Burst Time: 4
   - Priority: 1
   - Start Time: 6
   - Finish Time: 6 + 4 = 10
   - Waiting Time: Start Time - Arrival Time = 6 - 1 = 5

3. Process P4 (next highest priority) starts after P2 finishes:
   - Arrival Time: 3
   - Burst Time: 2
   - Priority: 2
   - Start Time: 10
   - Finish Time: 10 + 2 = 12
   - Waiting Time: Start Time - Arrival Time = 10 - 3 = 7

4. Process P3 (lowest priority) starts after P4 finishes:

- Arrival Time: 2
- Burst Time: 7
- Priority: 4
- Start Time: 12
- Finish Time: 12 + 7 = 19
- Waiting Time: Start Time - Arrival Time = 12 - 2 = 10

Calculating the Average Waiting Time:
Average Waiting Time = Total Waiting Time/Number of Processes
Total Waiting Time = 0 + 5 + 7 + 10 = 22
Number of Processes = 4
Average Waiting Time = 22/4 = 5.5
Therefore, the average waiting time for the given processes using Priority Scheduling is 5.5 units.

4. Consider the following processes with arrival times and burst times, and the time quantum for Round Robin scheduling is 2 units:

Process	Arrival Time	Burst Time
P1	0	4
P2	1	5
P3	2	2
P4	3	3

Calculate the average turnaround time using Round Robin scheduling.
Ans:-

Here's the given data:

Process	Arrival Time	Burst Time
P1	0	4
P2	1	5
P3	2	2
P4	3	3

Step-by-Step Calculation:

1. First Round (Time Quantum = 2 units):
   - P1 runs from 0 to 2 (Remaining Burst Time: 2)
   - P2 runs from 2 to 4 (Remaining Burst Time: 3)
   - P3 runs from 4 to 6 (Finished, Turnaround Time: 6 - 2 = 4)
   - P4 runs from 6 to 8 (Remaining Burst Time: 1)

2. Second Round (Time Quantum = 2 units):
   - P1 runs from 8 to 10 (Finished, Turnaround Time: 10 - 0 = 10)
   - P2 runs from 10 to 12 (Remaining Burst Time: 1)
   - P4 runs from 12 to 13 (Finished, Turnaround Time: 13 - 3 = 10)

3. Third Round (Time Quantum = 2 units):
   - P2 runs from 13 to 14 (Finished, Turnaround Time: 14 - 1 = 13)

Turnaround Times:
   - P1: 10 units
   - P2: 13 units
   - P3: 4 units
   - P4: 10 units

Calculating the Average Turnaround Time:
Average Turnaround Time = Total Turnaround Time/Number of Processes
Total Turnaround Time = 10 + 13 + 4 + 10 = 37
Number of Processes = 4
Average Turnaround Time = 37/4 = 9.25

Therefore, the average turnaround time for the given processes using Round Robin scheduling is 9.25 units.

5. Consider a program that uses the fork() system call to create a child process. Initially, the parent process has a variable x with a value of 5. After forking, both the parent and child processes increment the value of x by 1. What will be the final values of x in the parent and child processes after the fork() call?

Ans:
When the `fork()` system call is used to create a child process, the child process receives a copy of the parent's memory, including the variables. After the `fork()`, the parent and child processes run independently and do not share memory.

Initially, the parent process has a variable `x` with a value of 5. After the `fork()`, both the parent and child processes increment the value of `x` by 1.

Here's what happens:

1. Before `fork()`:
   - Parent process: `x = 5`

2. After `fork()`:
   - Parent process: `x = 5`
   - Child process: `x = 5` (copy of parent's `x`)

3. Both the parent and child processes increment `x` by 1:
   - Parent process: `x = 5 + 1 = 6`
   - Child process: `x = 5 + 1 = 6`

Therefore, after the `fork()` and increment operations, the final value of `x` in both the parent and child processes will be 6. Each process has its own independent copy of the variable `x`, and changes made in one process do not affect the other.