OS Assignment 2

********** Part A **********

What will the following commands do?

1. echo "Hello, World!"

⇨ print or display the Hello,World! as it is at output.

2. name="Productive"

⇨ this assign the string Productive to variable named as name.

3. touch file.txt

⇨ create only the blank text file.

4. ls -a

⇨ display all the present files and directories in present directory. includes hidden files also.

5. rm file.txt

⇨ remove or delete the file named as file.txt.

6. cp file1.txt file2.txt

⇨ copy the content of first file (file1.txt) and insert into file2.txt means second file.

7. mv file.txt /path/to/directory/

⇨ this command moves the file into the specific directory using the given path if directory is exist in it.

8. chmod 755 script.sh

⇨ this will change the permission of
7 => owner => read, write & execute permission
5 => group => read & execute permission
5 => other => read & execute permission

9. grep "pattern" file.txt

⇨ this will search for the line containing the same pattern name in the file and display that line.

10. kill PID

⇨ it used to stop the running process by its process ID.

11. mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt

⇨ in this command the second command will executed if only first one will get executed because use of logical and operator = &&.
first it create the directory name as mydir,

- then cd mydir => change the current directory to mydir,
- then touch file.txt => creates an empty text file inside the mydir,
- then echo “Hello, World!” > file.txt => it will print the string Hello, World inside the file.txt
- then cat file.txt => display the content inside the file.txt to the output.

12. ls -l | grep ".txt"

⇨ this will show the all files and directories with its information and print all the lines inside it containing .txt string or pattern.

13. cat file1.txt file2.txt | sort | uniq

⇨ prints the content of both file1 and file2, sort the lines in alphabetical order and removes the duplicate lines and print the uniq lines.

14. ls -l | grep "^d"

⇨ list all the files and directories and print the lines containing the string given inside double quotes.

15. grep -r "pattern" /path/to/directory/

⇨ this will check the string pattern inside the directory given in the path.

16. cat file1.txt file2.txt | sort | uniq –d

⇨ this will print the the content inside the file1 and file2 and sort it accordingly to the aplphabetical order and print the uniq duplicate line.

17. chmod 644 file.txt

⇨ this will change the permission of file to
6 => owner => read & write
4 => group => read permission
4 => other => read permission

18. cp -r source_directory destination_directory

⇨ this will copy all the content and subdirectories inside the source directory to the destination directory.

19. find /path/to/search -name "*.txt"

⇨ this command searches for the file with *.txt pattern inside the given path and directory and print the full path of the found file

20. chmod u+x file.txt

⇨ this will give the execution permission to the owner of the file.

21. echo $PATH

⇨ this will print the value of the variable having named PATH.

********** 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

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 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.

=> false

Identify the Incorrect Commands:

1. chmodx is used to change file permissions.

2. cpy is used to copy files and directories.

3. mkfile is used to create a new file.

4. catx is used to concatenate files.

5. rn is used to rename files.

=> All this command are incorrect because the correct command for the operation are :

1) chmod

2) cp

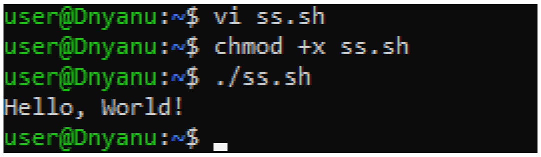
3) cat > or touch

4) cat

5) mv

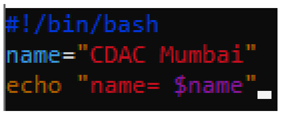
********** Part C **********

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

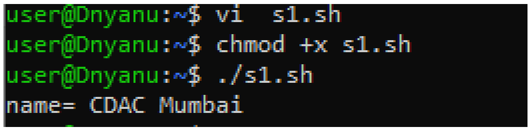


```
user@Dnyanu:~$ vi ss.sh
user@Dnyanu:~$ chmod +x ss.sh
user@Dnyanu:~$ ./ss.sh
Hello, World!
user@Dnyanu:~$
```

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

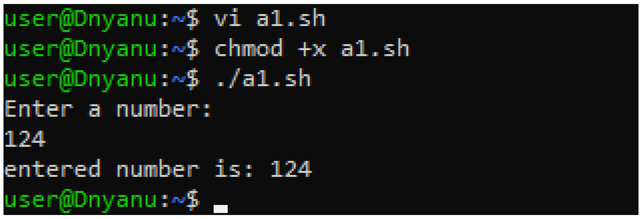


```
#!/bin/bash
name="CDAC Mumbai"
echo "name= $name"
```



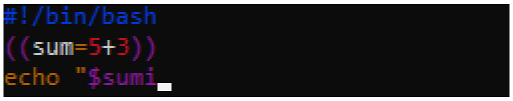
```
user@Dnyanu:~$ vi  s1.sh
user@Dnyanu:~$ chmod +x s1.sh
user@Dnyanu:~$ ./s1.sh
name= CDAC Mumbai
```

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

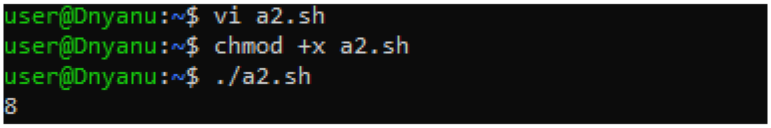


```
user@Dnyanu:~$ vi a1.sh
user@Dnyanu:~$ chmod +x a1.sh
user@Dnyanu:~$ ./a1.sh
Enter a number:
124
entered number is: 124
user@Dnyanu:~$
```

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

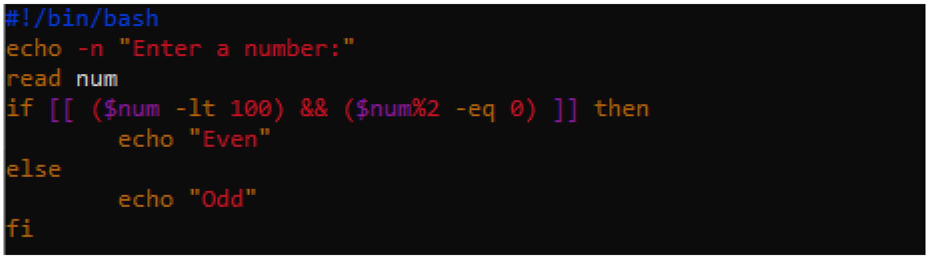


```
#!/bin/bash
((sum=5+3))
echo "$sumi
```

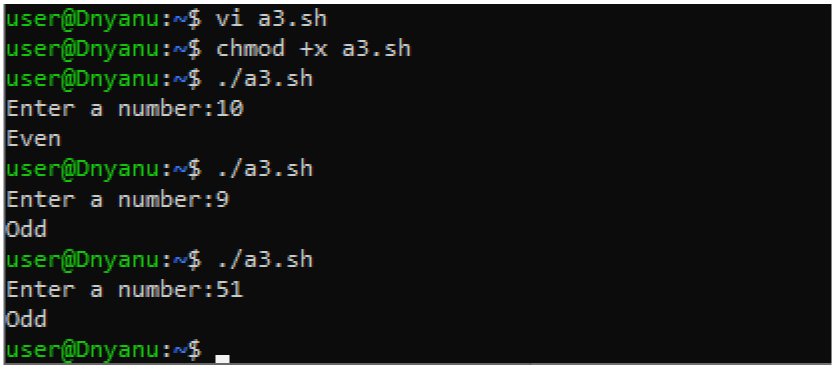


```
user@Dnyanu:~$ vi a2.sh
user@Dnyanu:~$ chmod +x a2.sh
user@Dnyanu:~$ ./a2.sh
8
```

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

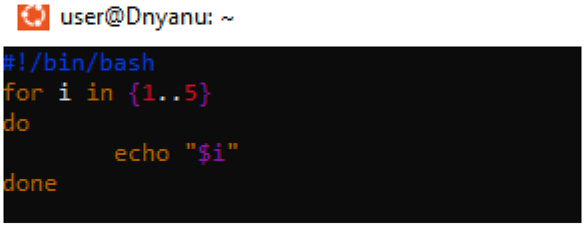


```
#!/bin/bash
echo -n "Enter a number:"
read num
if [[ ($num -lt 100) && ($num%2 -eq 0) ]] then
        echo "Even"
else
        echo "Odd"
fi
```

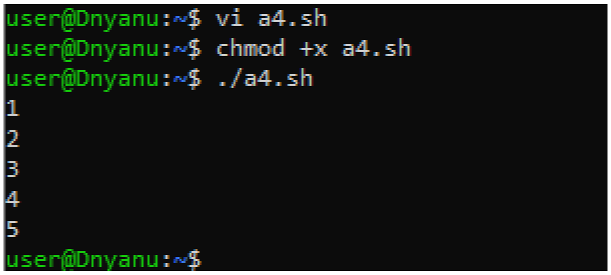


```
user@Dnyanu:~$ vi a3.sh
user@Dnyanu:~$ chmod +x a3.sh
user@Dnyanu:~$ ./a3.sh
Enter a number:10
Even
user@Dnyanu:~$ ./a3.sh
Enter a number:9
Odd
user@Dnyanu:~$ ./a3.sh
Enter a number:51
Odd
user@Dnyanu:~$ 
```

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

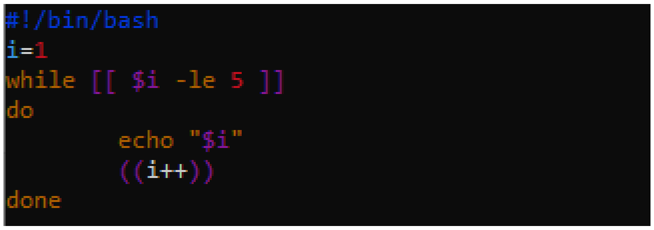


```
#!/bin/bash
for i in {1..5}
do
        echo "$i"
done
```

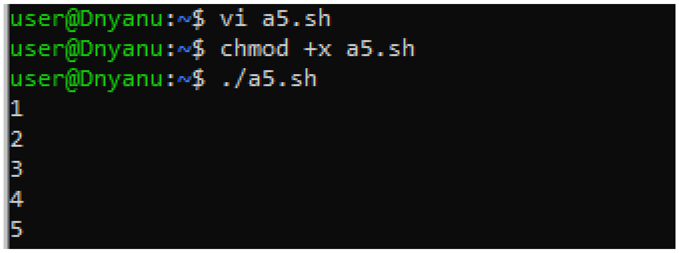


```
user@Dnyanu:~$ vi a4.sh
user@Dnyanu:~$ chmod +x a4.sh
user@Dnyanu:~$ ./a4.sh
1
2
3
4
5
user@Dnyanu:~$
```

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

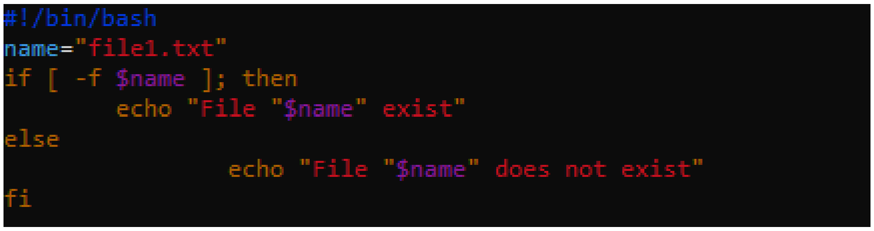


```
#!/bin/bash
i=1
while [[ $i -le 5 ]]
do
        echo "$i"
        ((i++))
done
```

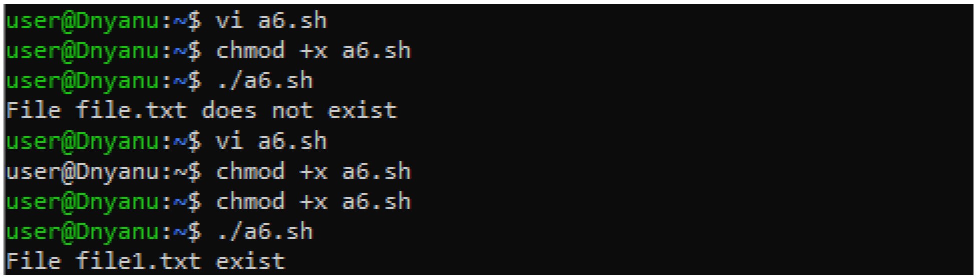


```
user@Dnyanu:~$ vi a5.sh
user@Dnyanu:~$ chmod +x a5.sh
user@Dnyanu:~$ ./a5.sh
1
2
3
4
5
```

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".

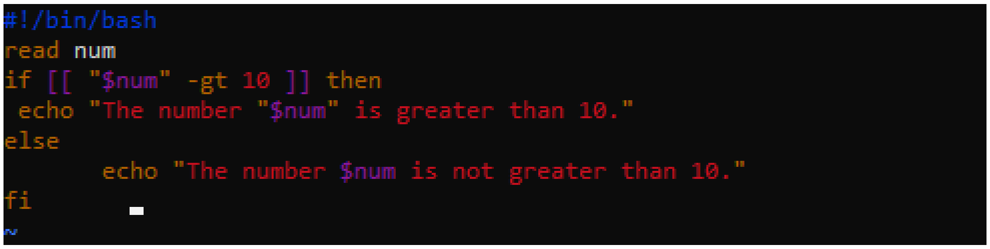


```
#!/bin/bash
name="file1.txt"
if [ -f $name ]; then
        echo "File "$name" exist"
else
                echo "File "$name" does not exist"
fi
```

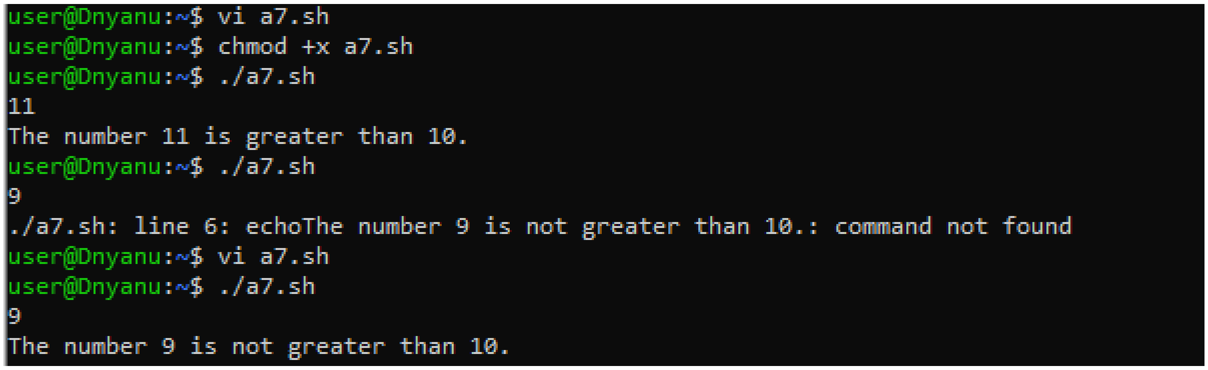


```
user@Dnyanu:~$ vi a6.sh
user@Dnyanu:~$ chmod +x a6.sh
user@Dnyanu:~$ ./a6.sh
File file.txt does not exist
user@Dnyanu:~$ vi a6.sh
user@Dnyanu:~$ chmod +x a6.sh
user@Dnyanu:~$ chmod +x a6.sh
user@Dnyanu:~$ ./a6.sh
File file1.txt exist
```

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

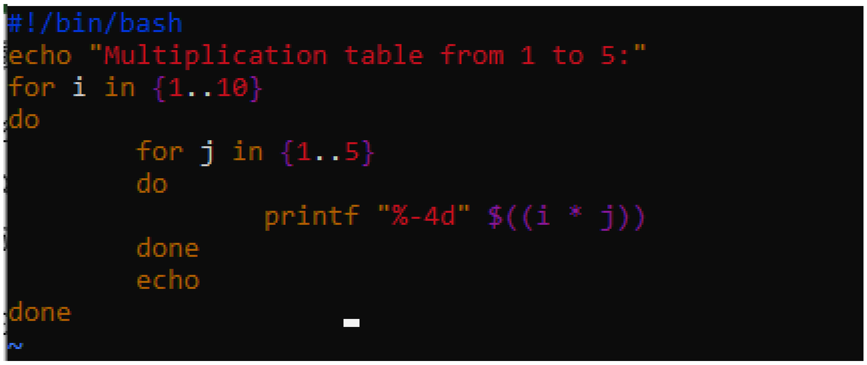


```
#!/bin/bash
read num
if [[ "$num" -gt 10 ]] then
 echo "The number "$num" is greater than 10."
else
       echo "The number $num is not greater than 10."
fi
~
```

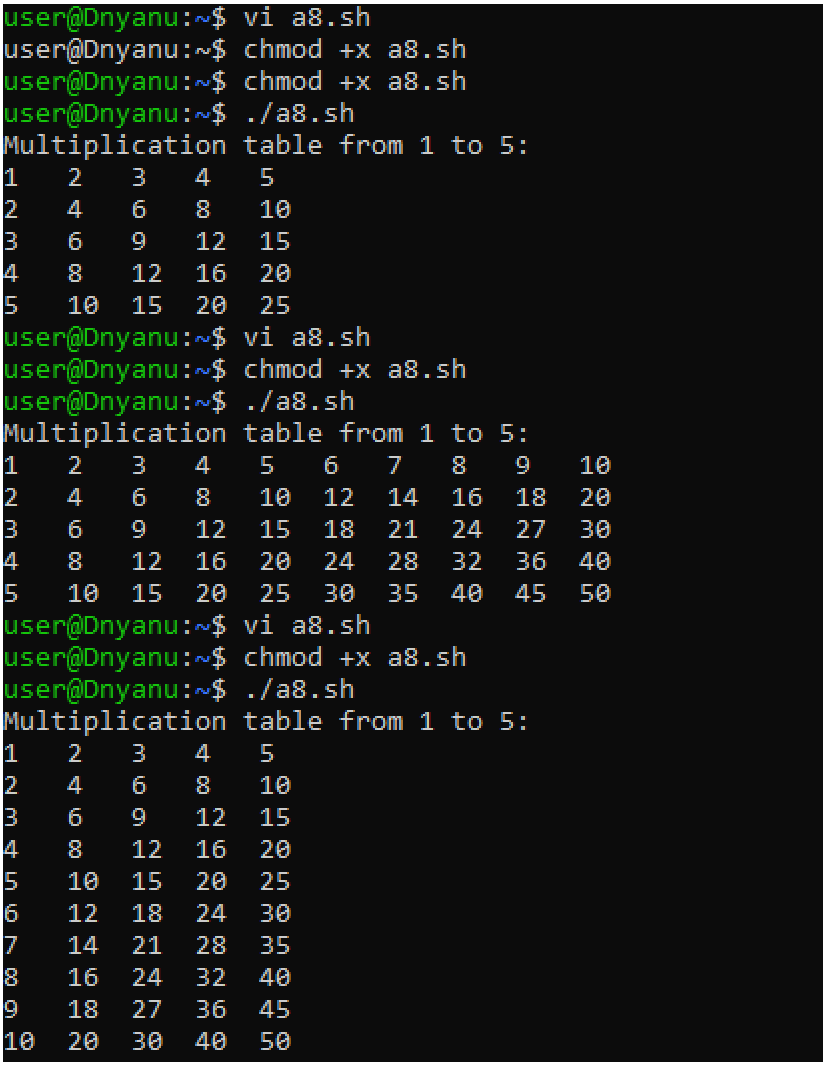


```
user@Dnyanu:~$ vi a7.sh
user@Dnyanu:~$ chmod +x a7.sh
user@Dnyanu:~$ ./a7.sh
11
The number 11 is greater than 10.
user@Dnyanu:~$ ./a7.sh
9
./a7.sh: line 6: echoThe number 9 is not greater than 10.: command not found
user@Dnyanu:~$ vi a7.sh
user@Dnyanu:~$ ./a7.sh
9
The number 9 is not greater than 10.
```

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.

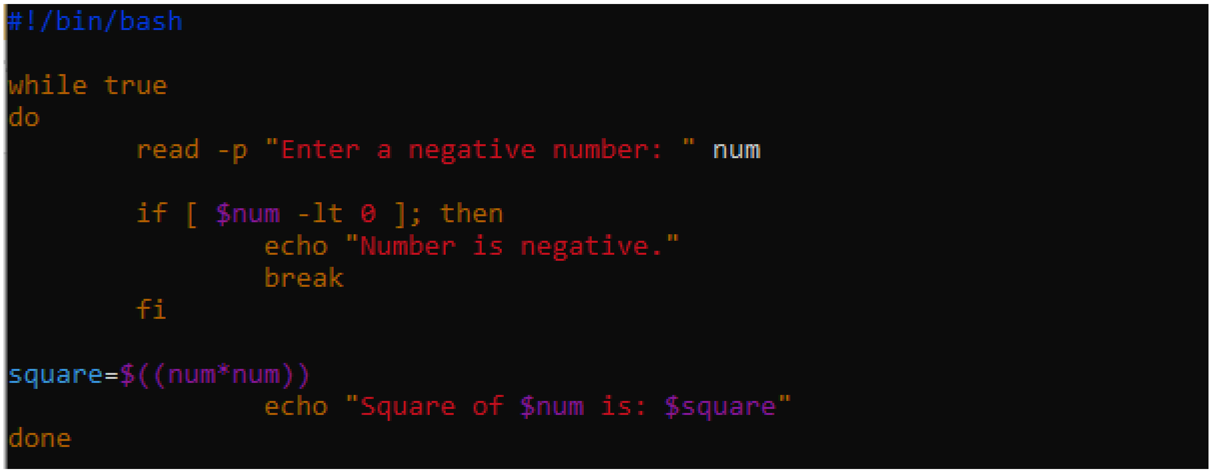


```
#!/bin/bash
echo "Multiplication table from 1 to 5:"
for i in {1..10}
do
        for j in {1..5}
        do
                printf "%-4d" $((i * j))
        done
        echo
done
~
```

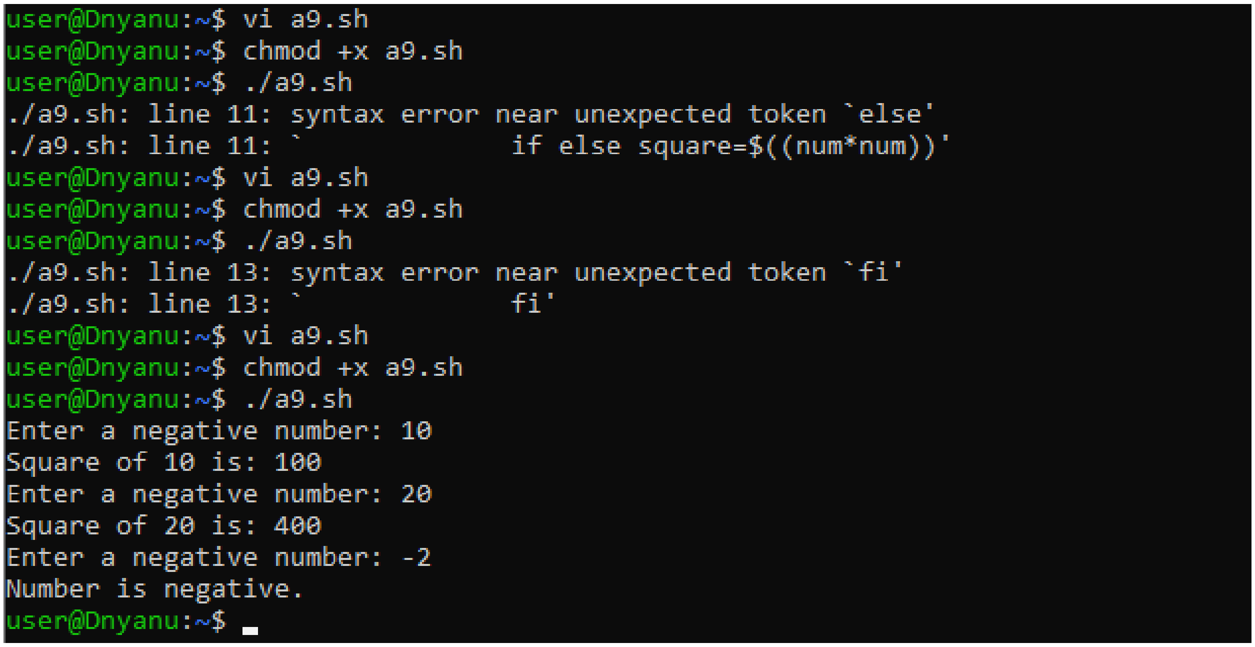


```
user@Dnyanu:~$ vi a8.sh
user@Dnyanu:~$ chmod +x a8.sh
user@Dnyanu:~$ chmod +x a8.sh
user@Dnyanu:~$ ./a8.sh
Multiplication table from 1 to 5:
1    2    3    4    5
2    4    6    8    10
3    6    9    12   15
4    8    12   16   20
5    10   15   20   25
user@Dnyanu:~$ vi a8.sh
user@Dnyanu:~$ chmod +x a8.sh
user@Dnyanu:~$ ./a8.sh
Multiplication table from 1 to 5:
1    2    3    4    5    6    7    8    9    10
2    4    6    8    10   12   14   16   18   20
3    6    9    12   15   18   21   24   27   30
4    8    12   16   20   24   28   32   36   40
5    10   15   20   25   30   35   40   45   50
user@Dnyanu:~$ vi a8.sh
user@Dnyanu:~$ chmod +x a8.sh
user@Dnyanu:~$ ./a8.sh
Multiplication table from 1 to 5:
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
```

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.

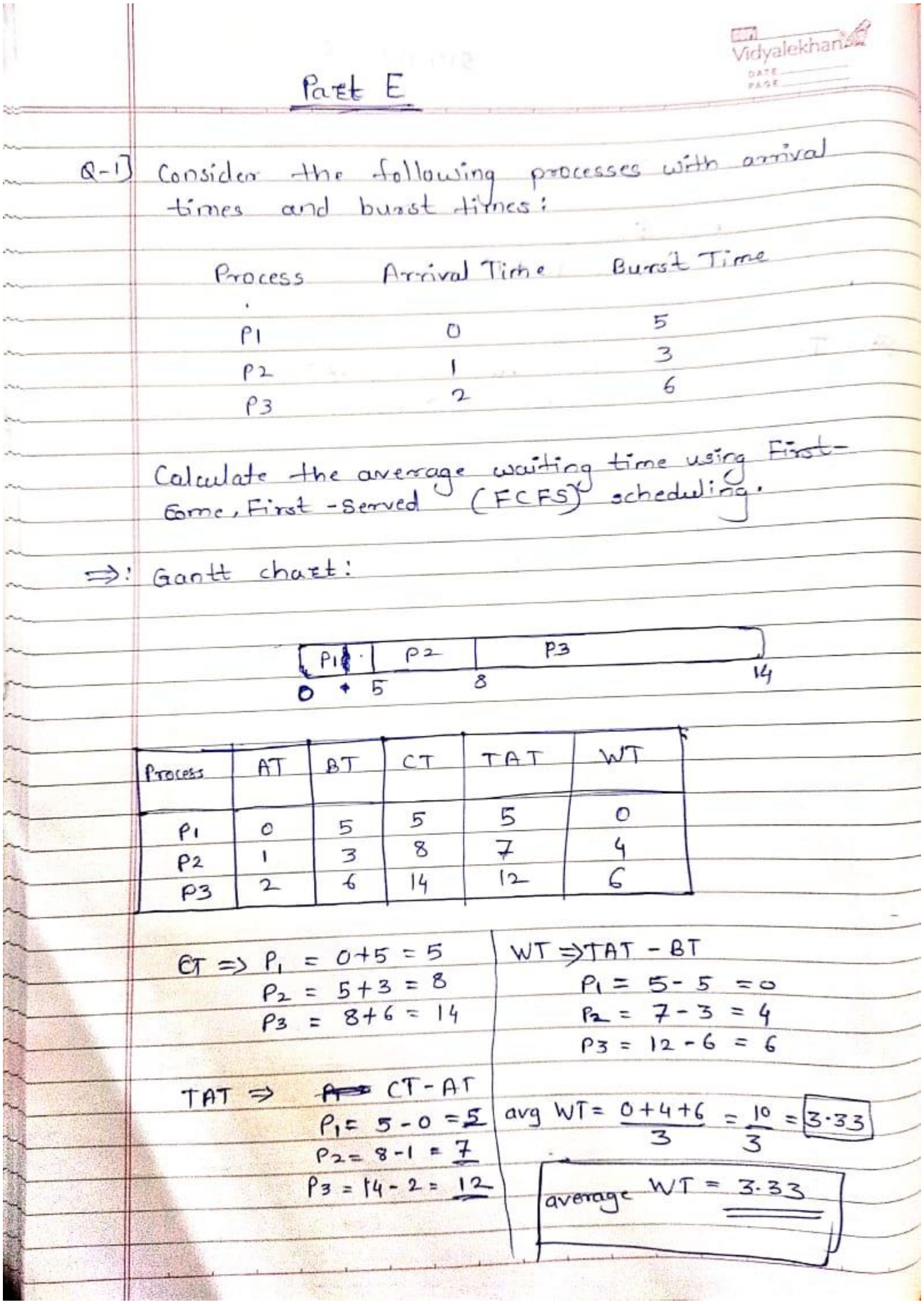


```
#!/bin/bash

while true
do
        read -p "Enter a negative number: " num

        if [ $num -lt 0 ]; then
                echo "Number is negative."
                break
        fi

square=$((num*num))
                echo "Square of $num is: $square"
done
```



```
user@Dnyanu:~$ vi a9.sh
user@Dnyanu:~$ chmod +x a9.sh
user@Dnyanu:~$ ./a9.sh
./a9.sh: line 11: syntax error near unexpected token `else'
./a9.sh: line 11: `             if else square=$((num*num))'
user@Dnyanu:~$ vi a9.sh
user@Dnyanu:~$ chmod +x a9.sh
user@Dnyanu:~$ ./a9.sh
./a9.sh: line 13: syntax error near unexpected token `fi'
./a9.sh: line 13: `             fi'
user@Dnyanu:~$ vi a9.sh
user@Dnyanu:~$ chmod +x a9.sh
user@Dnyanu:~$ ./a9.sh
Enter a negative number: 10
Square of 10 is: 100
Enter a negative number: 20
Square of 20 is: 400
Enter a negative number: -2
Number is negative.
user@Dnyanu:~$
```

********** **Part E** **********



Part E

Q-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.

⇒: Gantt chart:

P1
P2
P3
0
5
8
14

Process	AT	BT	CT	TAT	WT
P1	0	5	5	5	0
P2	1	3	8	7	4
P3	2	6	14	12	6

CT ⇒ $P_1 = 0+5 = 5$

$P_2 = 5+3 = 8$

$P_3 = 8+6 = 14$

TAT ⇒ ~~AT~~ CT - AT

$P_1 = 5 - 0 = \underline{5}$

$P_2 = 8 - 1 = \underline{7}$

$P_3 = 14 - 2 = \underline{\underline{12}}$

WT ⇒ TAT - BT

$P_1 = 5 - 5 = 0$

$P_2 = 7 - 3 = 4$

$P_3 = 12 - 6 = 6$

avg WT = $\frac{0+4+6}{3} = \frac{10}{3} = \boxed{3.33}$

$\boxed{\text{average WT} = \underline{\underline{3.33}}}$

Q-2] Consider the following processes with arrival times & burst times:

Process	AT	BT	CT	TAT (CT-AT)
P1	0	3	3	$P_1 = 3 - 0 = 3$
P2	1	5	13	$P_2 = 13 - 1 = 12$
P3	2	1	4	$P_3 = 4 - 2 = 2$
P4	3	4	8	$P_4 = 8 - 3 = 5$
				avg = 5.5

Calculate the average turnaround time Shortest Job First (SJF) scheduling.

⇒:

CT ⇒ $P_1 \Rightarrow 0+3 \Rightarrow 3$

Burst time of $P_3 < P_2$ & P_4

$P_3 \Rightarrow 3 + 1 \Rightarrow 4$

Burst time of $P_4 < P_2$ (∵ P_2 is waiting)

$P_4 \Rightarrow 4 + 4 \Rightarrow 8$

Now, only P_2 remains,

$P_2 \Rightarrow 8 + 5 \Rightarrow 13$

TAT ⇒ ~~P~~ CT - AT

$P_1 = 3 - 0 = 3$

$P_2 = 13 - 1 = 12$

$P_3 = 4 - 2 = 2$

$P_4 = 8 - 3 = 5$

$$\text{Average TAT} = \frac{3+12+2+5}{4} = \frac{22}{4} = \boxed{5.5}$$

∴ Average TAT = 5.5 time unit

Q-3] Consider the following processes with arrival times, burst times, + priorities (lower number indicates higher priority):

Process	AT	BT	Priority	CT	TAT	WT
P1	0	6	3	6	6	0
P2	1	4	1	10	9	5
P3	2	7	4	19	17	10
P4	3	2	2	12	9	7
						avg = 5.5

Calculate average WT using priority algorithm.

⇒ first P1 enters (arrive) then according to less number of priority gets higher priority for execution.

CT

$P_1 \Rightarrow 0 - 6 = 6$

$P_2 \Rightarrow 6 + 4 = 10$

$P_4 \Rightarrow 10 + 2 = 12$

$P_3 \Rightarrow 12 + 7 = 19$

TAT ⇒ CT − AT

$P_1 = 6 - 0 = 6$

$P_2 = 10 - 1 = 9$

$P_3 = 19 - 2 = 17$

$P_4 = 12 - 3 = 9$

WT = (TAT − BT)

$P_1 = 6 - 6 = 0$

$P_2 = 9 - 4 = 5$

$P_3 = 17 - 7 = 10$

$P_4 = 9 - 2 = 7$

$$\text{Average waiting time} = \frac{0 + 5 + 10 + 7}{4}$$

$$= \frac{22}{4}$$

$$\boxed{\text{avg } WT = 5.5}$$

Q-4] Consider the following processes with arrival times & bursts times, and the time quantum for Round Robin scheduling is 2 units:

Process	AT	BT	CT	TAT = (CT-AT)
P1	0	4	8	8
P2	1	5	14	13
P3	2	2	6	4
P4	3	3	13	10
				avg TAT = 8.7

Calculate the averrage turnaround time using Round Robin scheduling.

1 = 4-2 = 2-2 = 0
P2 = 5-2 = 3-2 = 1
P3 = 2-2 = 0
P4 = 3-2 = 1

P4 = 1

P1	P2	P3	P41.	P1	P2	P4	P2

0 2 4 6 8 10 12 13 14

P1	P2	P3	P1	P4	P2	P4	P2

0 2 4 6 8 10 12 13 14

TAT = CT - AT

$P_1 = 8 - 0 = 8$

$P_2 = 14 - 1 = 13$

$P_3 = 6 - 2 = 4$

$P_4 = 13 - 3 = 10$

average TAT (turn around time) = $\frac{8 + 13 + 4 + 10}{4}$

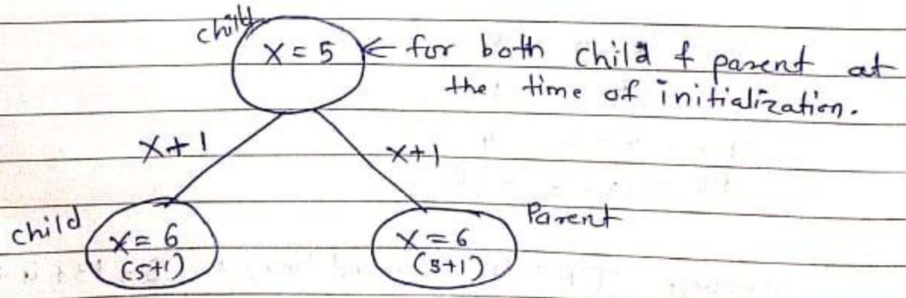
$= \frac{35}{4} = 8.7$

Q-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 & child processes ~~after~~ 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?

⇒:

- When a fork() calls & create a child process it will copy the entire memory of parent process as it is.
- It does not affect on each of them both contain private copy of X = 5
- Change in child's copy do not affect the parent copy and same for vice, versa.
- At final

∴ Parent process = 5+1 ⇒ $\boxed{X = 6}$
Child process = 5+1 ⇒ $\boxed{X = 6}$


child
X = 5
← for both child & parent at the time of initialization.
X+1
X+1
child
X= 6
(5+1)
Parent
X=6
(5+1)