Looping

1. Write a Bash script that uses a for loop to print the numbers from 1 to 10.

```
#!/bin/bash
for ((i=1; i<=10; i++))
do
echo "$i"
done
```

```
┌──(nisha㉿kali)-[~/bash]
└─$ ./1.sh
1
2
3
4
5
6
7
8
9
10
```

2. Create a Bash script that uses a while loop to print the even numbers from 2 to 20.

```
#!/bin/bash
num=2
while [ $num -le 20 ]
do
echo $num
num=$((num + 2 ))
done
```

```
┌──(nisha㉿kali)-[~/bash]
└─$ ./2.sh
2
4
6
8
10
12
14
16
18
20
```

3. Explain the difference between a for loop and a while loop in Bash scripting.
   For loops are frequently used with known, finite lists, such as a set of integers, a list of things, or a set of counters.
   While loops are handy for conditions without a known limit, they can also be utilized with lists. It can be used to execute commands when a condition is true.

4. Write a Bash script that uses a until loop to count down from 5 to 1.

```
1 #!/bin/bash
2 count=5
3 until [ $count -eq 0 ]
4 do
5 echo $count
6 count=$((count - 1 ))
7 done
```

```
┌──(nisha㉿kali)-[~/bash]
└─$ chmod +x 3.sh

┌──(nisha㉿kali)-[~/bash]
└─$ ./3.sh
5
4
3
2
1

┌──(nisha㉿kali)-[~/bash]
└─$
```

Asking Input:

1. How do you prompt a user for their name in a Bash script and store it in a variable?

```
1 #!/bin/bash
2 echo "enter your name:"
3 read name
4 echo "Hi! $name. What's Up?"
```

```
┌──(nisha㉿kali)-[~/bash]
└─$ chmod +x 4.sh

┌──(nisha㉿kali)-[~/bash]
└─$ ./4.sh
enter your name:
nisha
Hi! nisha. What's Up?
```

2. Write a Bash script that asks the user for their age and then displays a message based on whether they are old enough to vote (18 or older).

```
1 #!/bin/bash
2 read -p "enter a age: " age
3 if ((age ⩾ 18)); then
4 echo "you are eligible to vote"
5 else
6 echo "sorry you are not eligible to vote"
7 fi
```

```
┌──(nisha㉿kali)-[~/bash]
└─$ ./6.sh
enter a age: 18
you are eligible to vote

┌──(nisha㉿kali)-[~/bash]
└─$ ./6.sh
enter a age: 15
sorry you are not eligible to vote

┌──(nisha㉿kali)-[~/bash]
└─$
```

Case Conditional Statement:

1. Explain the purpose of the case statement in Bash scripting.

   In bash scripts, a case statement is used when an option choice must be made amongst many options. Case statements offer an advantage over if-statements because they increase code readability and are easier to maintain. Case statements in Bash scripts are very similar to Case expressions in C.

2. Create a Bash script that uses a case statement to determine if a given input is a vowel or a consonant.

```
#!/bin/bash
read -p "Enter any letter: " letter
case "$letter" in
[aeiouAEIOU])
echo "$letter is a vowel."
;;
[bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ])
echo "$letter is a consonant."
;;
*)
echo "Invalid input "
;;
esac
```

```
┌──(nisha㉿kali)-[~/bash]
└─$ ./7.sh
Enter any letter: n
n is a consonant.

┌──(nisha㉿kali)-[~/bash]
└─$ ./7.sh
Enter any letter: i
i is a vowel.

┌──(nisha㉿kali)-[~/bash]
└─$ ./7.sh
Enter any letter: hg
Invalid input
```

3. What is the significance of the ;; in a case statement?
   The ;; marks the end of a case block in a Bash case statement. It's critical for indicating when one case block finishes and another begins.

Conditional Statements:

1. Write a Bash script that uses “if statement” to check if a file named "example.txt" exists in the current directory.

```bash
#!/bin/bash
file="example.txt"
if [ -e "$file" ]; then
echo "$file exists in the current directory."
else
echo "$file doesn't exists in the current directory."
fi
```

```
┌──(nisha㉿kali)-[~/bash]
└─$ ./8.sh
example.txt doesn't exists in the current directory.

┌──(nisha㉿kali)-[~/bash]
└─$ touch example.txt

┌──(nisha㉿kali)-[~/bash]
└─$ ls
1.sh  2.sh  3.sh  4.sh  5.sh  6.sh  7.sh  8.sh  backup.sh  cleanup.sh  example.txt  n  nishaa.sh  nisha.sh  pandey.sh

┌──(nisha㉿kali)-[~/bash]
└─$ ./8.sh
example.txt exists in the current directory.
```

2. How do you use the -eq operator to compare two numbers in a Bash if statement?

```bash
#!/bin/bash
read -p "enter a number: " num1
read -p "enter another number: " num2
if [ $num1 -eq $num2 ]; then
echo "Two numbers are equal"
else
echo " Two numbers aren't equal"
fi
```

```
┌──(nisha㉿kali)-[~/bash]
└─$ chmod +x 9.sh

┌──(nisha㉿kali)-[~/bash]
└─$ ./9.sh
enter a number: 4
enter another number: 4
Two numbers are equal

┌──(nisha㉿kali)-[~/bash]
└─$ ./9.sh
enter a number: 2
enter another number: 4
 Two numbers aren't equal
```

3. Write a Bash script that compares two numbers entered by the user and displays whether the first number is greater, smaller, or equal to the second number.

```
#!/bin/bash
read -p "enter a number: " num1
read -p "enter another number: " num2

if [ $num1 -gt $num2 ]; then
        echo "$num1 is greater than $num2."
elif [ $num1 -lt $num2 ]; then
        echo " $num1 is smaller than $num2. "
else
        echo "$num1 is equal to $num2. "
fi
```

```
┌──(nisha㉿kali)-[~/bash]
└─$ ./10.sh
enter a number: 1
enter another number: 2
 1 is smaller than 2.

┌──(nisha㉿kali)-[~/bash]
└─$ ./10.sh
enter a number: 2
enter another number: 1
2 is greater than 1.

┌──(nisha㉿kali)-[~/bash]
└─$ ./10.sh
enter a number: 21
enter another number: 21
21 is equal to 21.
```

Arithmetic:

1. Explain the purpose of the expr command in Bash scripting.

   For evaluating and carrying out various operations on expressions in Bash scripting, use the 'expr' command. The main function is to handle logical, string, and arithmetic operations in shell scripts.

2. Create a Bash script that calculates the sum of two numbers entered by the user.

```
#!/bin/bash
read -p "enter a number: " num1
read -p "enter another number: " num2
sum=$(( $num1 + $num2))
echo "The sum of two number is $sum."
```

```
┌──(nisha㉿kali)-[~/bash]
└─$ chmod +x 11.sh

┌──(nisha㉿kali)-[~/bash]
└─$ ./11.sh
enter a number: 3
enter another number: 2
The sum of two number is 5.

┌──(nisha㉿kali)-[~/bash]
└─$
```

3. How do you use the let command to perform arithmetic operations in Bash?

```
#!/bin/bash

read -p "enter a number: " num1
read -p "enter another number: " num2

let sum=$(( $num1 + $num2))
let sub=$(( $num1 - $num2))
let multi=$(( $num1 * $num2))
let div=$(( $num1 / $num2))
        echo "addition of two number is $sum"
        echo "subtraction of two number is $sub"
        echo "multiplication of two number is $multi"
        echo "division of two number is $div"
```

```
┌──(nisha㉿kali)-[~/bash]
└─$ bash 12.sh
enter a number: 2
enter another number: 1
addition of two number is 3
subtraction of two number is 1
multiplication of two number is 2
division of two number is 2
```

Operators

1. What is the purpose of the && operator in Bash scripting? Provide an example.

   The && operator is used in Bash scripting to execute commands only when certain conditions are met. It only permits to issue a further command if the initial one is successful. The second command is not executed if the first command fails. When writing a series of instructions, the && operator is frequently used to make each operation reply on the success of the one before it.
   Example:

```
1 #!/bin/bash
2 file="nisha.txt"
3 echo "hellooo! How are you?" > "$file" && cat "$file"
```

```
┌──(nisha㉿kali)-[~/bash]
└─$ bash 14.sh
hellooo! How are you?

┌──(nisha㉿kali)-[~/bash]
└─$ ls
10.sh  12.sh  14.sh  2.sh  4.sh  6.sh  8.sh  backup.sh   n          nisha.sh   pandey.sh
11.sh  13.sh  1.sh   3.sh  5.sh  7.sh  9.sh  cleanup.sh  nishaa.sh  nisha.txt
```

2. Write a Bash script that uses the -gt operator to check if a number is greater than 10.

```
#!/bin/bash
read -p "enter a number: " num

if [ $num -gt 10 ]; then
echo "$num is greater than 10."
else
echo "$num is not greater than 10"
fi
```

```
┌──(nisha㉿kali)-[~/bash]
└─$ bash 13.sh
enter a number: 9
9 is not greater than 10

┌──(nisha㉿kali)-[~/bash]
└─$ bash 13.sh
enter a number: 21
21 is greater than 10.
```

3. Explain the difference between = and == in Bash for string comparisons.

   Basic string comparisons are performed with the = operator, frequently in if statements or other conditional statements. It checks the equality of two strings. It is advised for straightforward, accurate string equality checks.
   And To match patterns and glob, use the == operator. Typically, it appears in places like case statements. For pattern-based string comparisons, it offers more freedom.

Mixed Questions:

1. Create a Bash script that asks the user for a number and then uses a loop to print the multiplication table for that number from 1 to 10.

```
#!/bin/bash
read -p "enter a number: " num
for i in $(seq 1 1 10)
do
let "mult =$num * $i"
echo "$num * $i = $mult"
done
```

```
┌──(nisha㉿kali)-[~/bash]
└─$ bash 17.sh
enter a number: 3
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30
```

2. Write a Bash script that takes a user's age as input and then uses a case statement to display a message based on whether they are a child, teenager, adult, or senior citizen.

```
#!/bin/bash
read -p "Enter your age: " age
case 1 in
        $(($age <= 12))
                echo "You are a child."
                ;;
        $(($age <= 19))
                echo "you are a teenager. "
                ;;
        $(($age <= 64))
                echo "you are adult."
                ;;
        *)
                echo "you are a senior citizen."
                ;;
esac
```


┌──(nisha㉿kali)-[~/bash]
└─$ bash 15.sh
Enter your age: 2
You are a child.
┌──(nisha㉿kali)-[~/bash]
└─$ bash 15.sh
Enter your age: 21
you are adult.
┌──(nisha㉿kali)-[~/bash]
└─$ bash 15.sh
Enter your age: 80
you are a senior citizen.