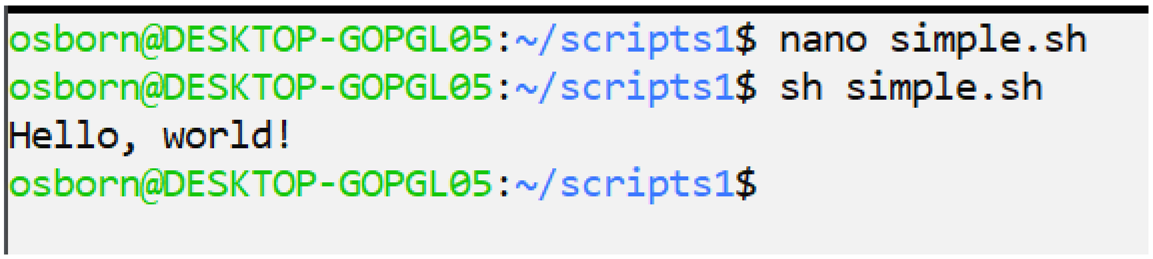
Assignment 9

a) Simple shell scripts,

Definition: Simple shell scripts are basic scripts written in shell (like Bash) that execute a sequence of commands or perform a specific task.

```
#!/bin/bash
echo "Hello, world!"
```

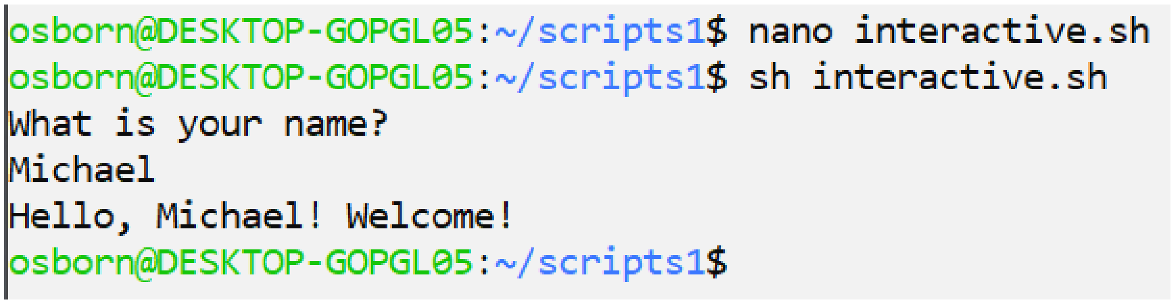


```
osborn@DESKTOP-GOPGL05:~/scripts1$ nano simple.sh
osborn@DESKTOP-GOPGL05:~/scripts1$ sh simple.sh
Hello, world!
osborn@DESKTOP-GOPGL05:~/scripts1$
```

b) Interactive shell script,

Definition: An interactive shell script prompts the user for input during execution and responds based on the provided input.

```
#!/bin/bash
echo "What is your name?"
read name
echo "Hello, $name! Welcome!"
```



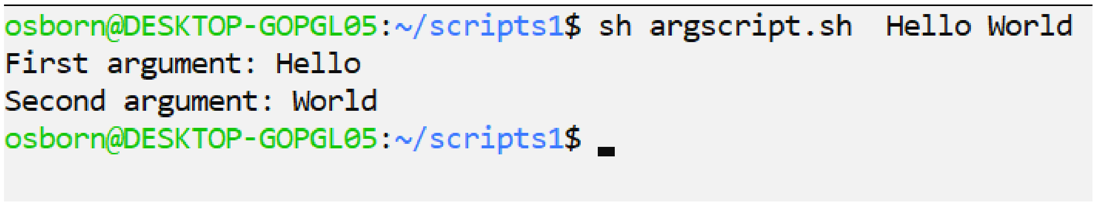
```
osborn@DESKTOP-GOPGL05:~/scripts1$ nano interactive.sh
osborn@DESKTOP-GOPGL05:~/scripts1$ sh interactive.sh
What is your name?
Michael
Hello, Michael! Welcome!
osborn@DESKTOP-GOPGL05:~/scripts1$
```

c) Using command line arguments,

Definition: Command line arguments are parameters passed to a shell script when it is executed, allowing customization and flexibility.

#!/bin/bash
echo "First argument: $1"
echo "Second argument: $2"

Terminal - ./script.sh arg1 arg2

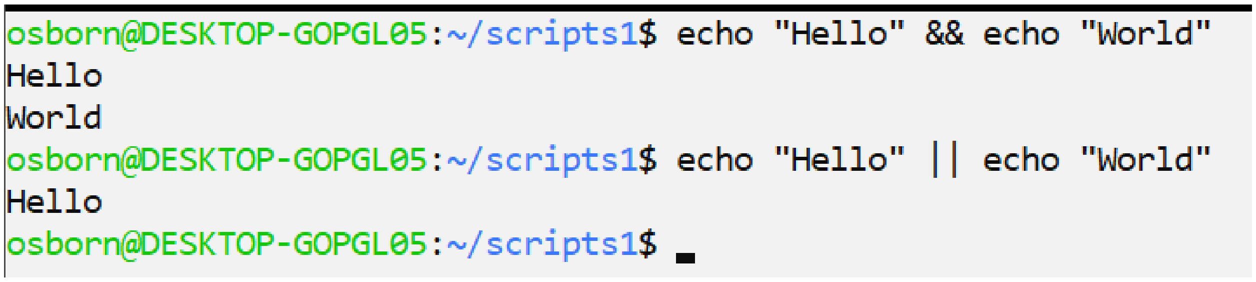


```
osborn@DESKTOP-GOPGL05:~/scripts1$ sh argscript.sh  Hello World
First argument: Hello
Second argument: World
osborn@DESKTOP-GOPGL05:~/scripts1$
```

d) Logical operator (&amp;&amp;, ||),

Definition: Logical operators (&& for AND, || for OR) are used for conditional execution of commands based on the success or failure of preceding commands.

echo "Hello" && echo "World"
echo "Hello" || echo "World"

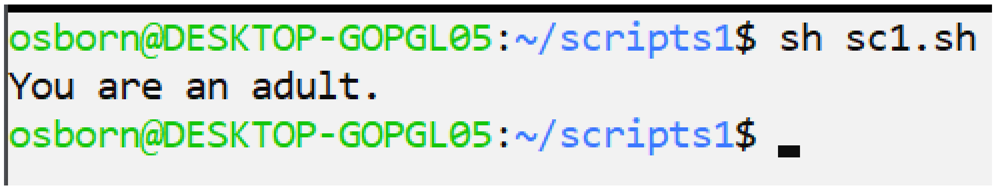


```
osborn@DESKTOP-GOPGL05:~/scripts1$ echo "Hello" && echo "World"
Hello
World
osborn@DESKTOP-GOPGL05:~/scripts1$ echo "Hello" || echo "World"
Hello
osborn@DESKTOP-GOPGL05:~/scripts1$
```

e) Condition checking (if-then, if-then-else-fi, if-then-elif-else-fi, case),

- Example (if-then):

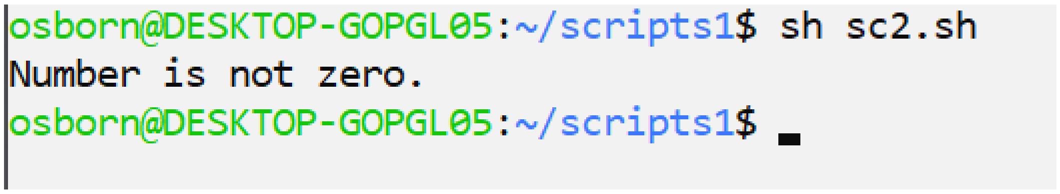
```
#!/bin/bash
# Simple if-then condition
age=20
if [ $age -ge 18 ]
then
   echo "You are an adult."
fi
```



```
osborn@DESKTOP-GOPGL05:~/scripts1$ sh sc1.sh
You are an adult.
osborn@DESKTOP-GOPGL05:~/scripts1$
```

- Example (if-then-else-fi):

```
#!/bin/bash
# If-then-else condition
num=5
if [ $num -eq 0 ]
then
   echo "Number is zero."
else
   echo "Number is not zero."
fi
```

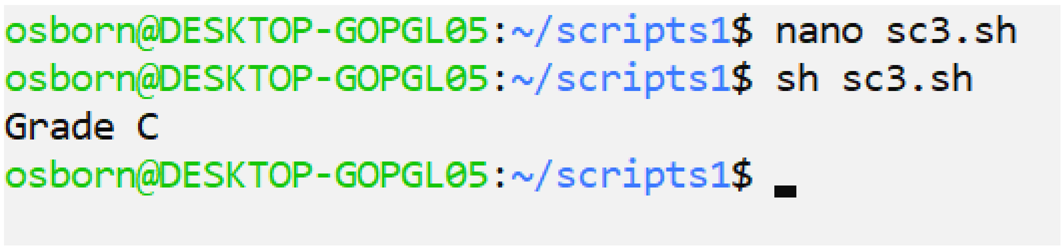


```
osborn@DESKTOP-GOPGL05:~/scripts1$ sh sc2.sh
Number is not zero.
osborn@DESKTOP-GOPGL05:~/scripts1$
```

- Example (if-then-elif-else-fi):

```
#!/bin/bash
# If-then-elif-else condition
score=75
if [ $score -ge 90 ]
then
   echo "Grade A"
elif [ $score -ge 80 ]
then
   echo "Grade B"
else
```

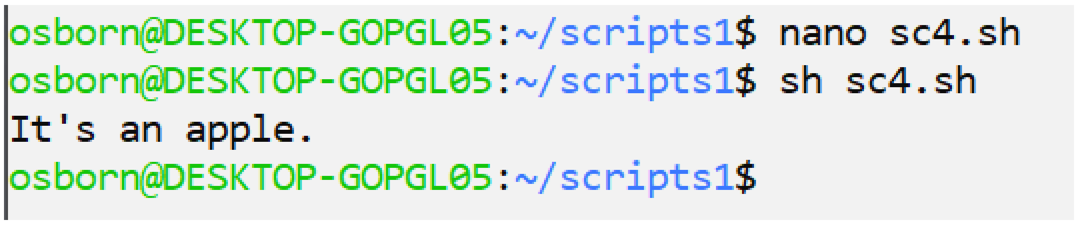
```
    echo "Grade C"
fi
```



```
osborn@DESKTOP-GOPGL05:~/scripts1$ nano sc3.sh
osborn@DESKTOP-GOPGL05:~/scripts1$ sh sc3.sh
Grade C
osborn@DESKTOP-GOPGL05:~/scripts1$
```

- Example (case):

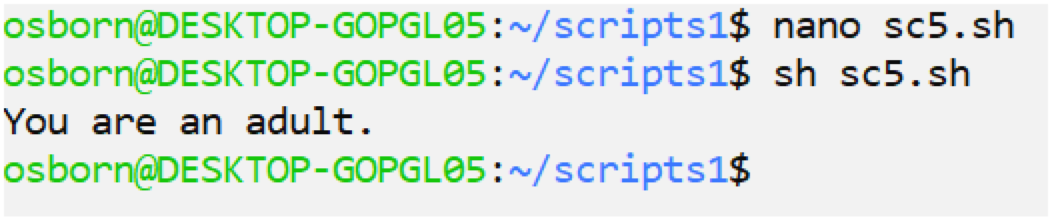
```
#!/bin/bash
# Case statement
fruit="apple"
case $fruit in
    apple)
        echo "It's an apple."
        ;;
    banana)
        echo "It's a banana."
        ;;
    *)
        echo "Unknown fruit."
        ;;
esac
```



```
osborn@DESKTOP-GOPGL05:~/scripts1$ nano sc4.sh
osborn@DESKTOP-GOPGL05:~/scripts1$ sh sc4.sh
It's an apple.
osborn@DESKTOP-GOPGL05:~/scripts1$
```

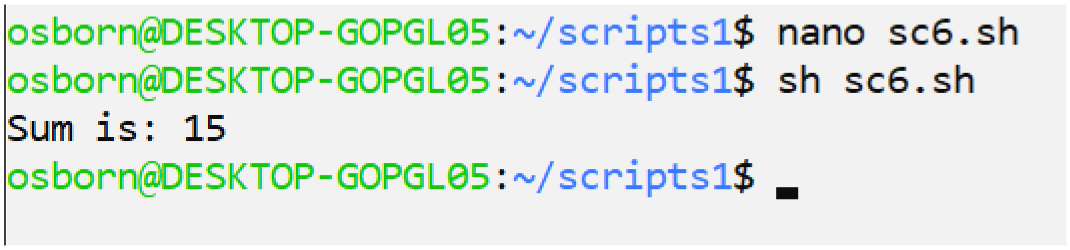
f) Expression evaluation (test, []),

```
#!/bin/bash
# Using test command for expression evaluation
age=25
if test $age -ge 18
then
    echo "You are an adult."
fi
```


```
osborn@DESKTOP-GOPGL05:~/scripts1$ nano sc5.sh
osborn@DESKTOP-GOPGL05:~/scripts1$ sh sc5.sh
You are an adult.
osborn@DESKTOP-GOPGL05:~/scripts1$
```

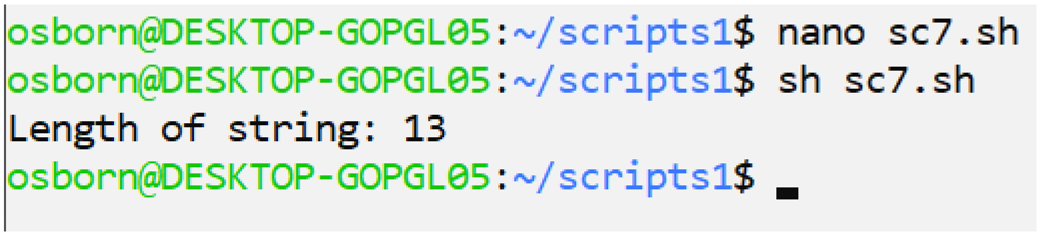
g) Computation (expr),

```
#!/bin/bash
# Using expr for computation
num1=10
num2=5
sum=$(expr $num1 + $num2)
echo "Sum is: $sum"
```


```
osborn@DESKTOP-GOPGL05:~/scripts1$ nano sc6.sh
osborn@DESKTOP-GOPGL05:~/scripts1$ sh sc6.sh
Sum is: 15
osborn@DESKTOP-GOPGL05:~/scripts1$
```

h) Using expr for strings,

```
#!/bin/bash
# Using expr for string operations
greeting="Hello, world!"
length=$(expr length "$greeting")
echo "Length of string: $length"
```


```
osborn@DESKTOP-GOPGL05:~/scripts1$ nano sc7.sh
osborn@DESKTOP-GOPGL05:~/scripts1$ sh sc7.sh
Length of string: 13
osborn@DESKTOP-GOPGL05:~/scripts1$
```

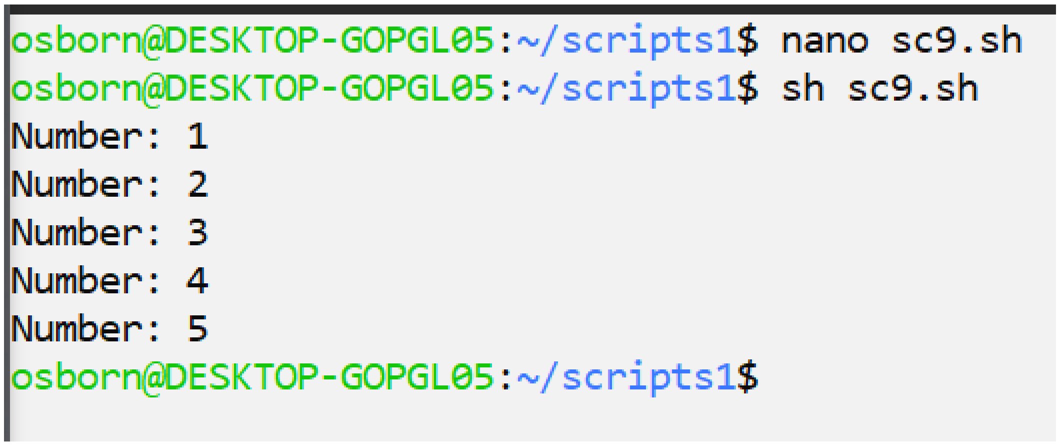
i) Loop (while, for, until, continue),

- Example (while loop):

Definition: The while loop executes a block of code as long as a specified condition is true.

```
#!/bin/bash
# While loop to count numbers from 1 to 5 using expr for incrementing

num=1
while [ $num -le 5 ]
do
    echo "Number: $num"
    num=$(expr $num + 1)
done
```



```
osborn@DESKTOP-GOPGL05:~/scripts1$ nano sc9.sh
osborn@DESKTOP-GOPGL05:~/scripts1$ sh sc9.sh
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
osborn@DESKTOP-GOPGL05:~/scripts1$
```

- Example (for loop):

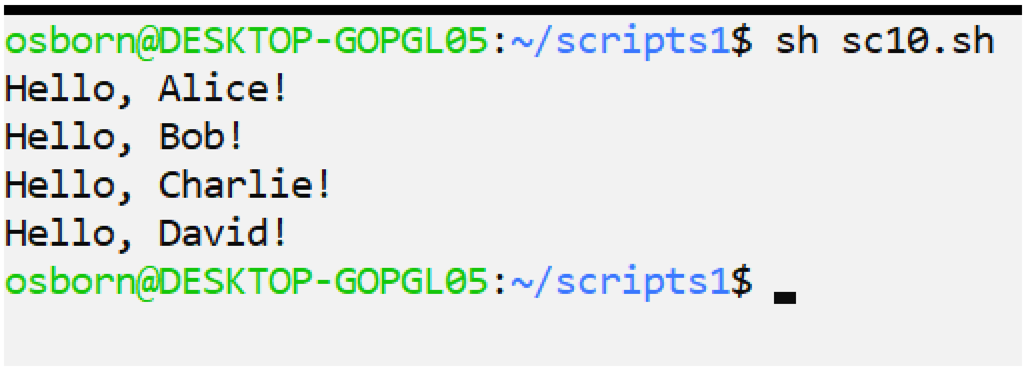
Definition: The for loop iterates over a sequence of values (such as numbers or items in a list) and executes a block of code for each value.

```
#!/bin/bash
# For loop to iterate over a list of names without using arrays

names="Alice Bob Charlie David"

for name in $names
do
    echo "Hello, $name!"
```

```
done
```



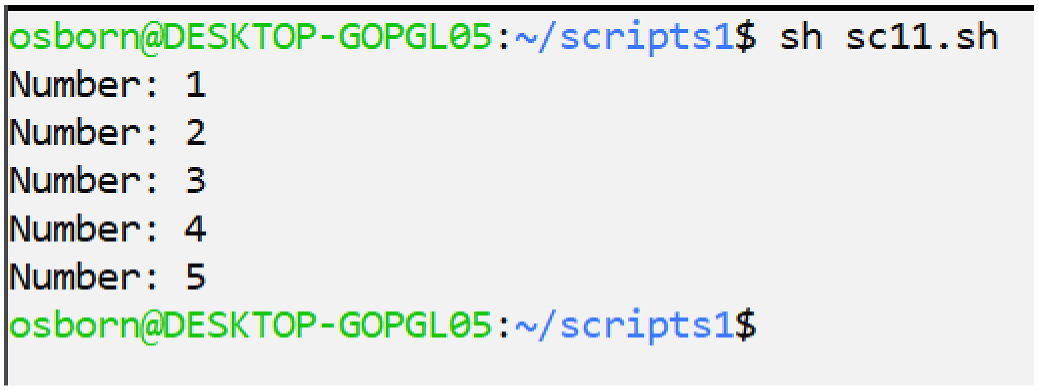
```
osborn@DESKTOP-GOPGL05:~/scripts1$ sh sc10.sh
Hello, Alice!
Hello, Bob!
Hello, Charlie!
Hello, David!
osborn@DESKTOP-GOPGL05:~/scripts1$
```

- Example (until loop):

Definition: The until loop executes a block of code as long as a specified condition is false.

```
#!/bin/bash
# Until loop to count numbers from 1 to 5 using expr for incrementing

num=1
until [ $num -gt 5 ]
do
    echo "Number: $num"
    num=$(expr $num + 1)
done
```



```
osborn@DESKTOP-GOPGL05:~/scripts1$ sh sc11.sh
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
osborn@DESKTOP-GOPGL05:~/scripts1$
```

- Example (continue):

Definition: The continue statement is used inside loops to skip the remaining commands within the current iteration and proceed to the next iteration

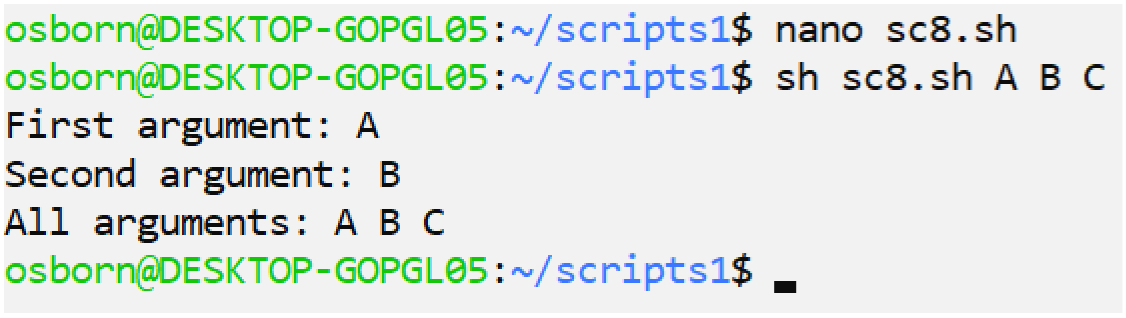
```
#!/bin/bash
# For loop to print odd numbers from 1 to 10 using continue

echo "Printing odd numbers from 1 to 10:"
for (( i = 1; i <= 10; i++ ))
do
   if (( i % 2 == 0 ))
   then
      continue  # Skip even numbers
   fi
   echo "Odd number: $i"
done
```

j) Use of positional parameters

Definition: Positional parameters refer to arguments passed to a script or function at runtime ($1, $2, etc.).

```
#!/bin/bash
# Using positional parameters
echo "First argument: $1"
echo "Second argument: $2"
echo "All arguments: $@"
```



```
osborn@DESKTOP-GOPGL05:~/scripts1$ nano sc8.sh
osborn@DESKTOP-GOPGL05:~/scripts1$ sh sc8.sh A B C
First argument: A
Second argument: B
All arguments: A B C
osborn@DESKTOP-GOPGL05:~/scripts1$
```