Node.js
1)Node js is an environment for building front end apps as well as backend using javascript as language.it is different from others because through it we can build both the apps
2)The event driven in node js allows it to handle many connections without consuming extra resources
It allows node js to respond quickly which enhances the responsiveness of applications
3)asynchronous functions are handled through three ways:
Call back hells
Promises
async/await
Advanced javascript
1)
1)Scope:

```
//var is global
var firstName="Mehreen"
function printName(){
    console.log(firstName)
}
printName()
console.log(firstName)

//output
Mehreen
Mehreen
if(1===1){
    var lastName="mehreen"
    console.log(lastName)
}
console.log(lastName)
//output
Fatima
fatima
```

Here we can see that the variable that is declared in the global space is accessed locally and globally.var is not blocked scope by that I mean is the second piece of code that is showing that if var is called outside the block will not show error but the output.
let is global and block scoped

```
let firstName="Mehreen"
function printName(){
    console.log(firstName)


}
```

```
printName()
console.log(firstName)
if(1===1){
    let lastName="fatima"
    console.log(lastName)
}
console.log(lastName)
//output
Mehreen
Mehreen
fatima
d:\Desktop\code girls\phase 2\practise\class lessons\day2.js:44
console.log(lastName)
            ^

ReferenceError: lastName is not defined
```

Const is also global and block scoped
2)Redeclaration and Reassigning:
Var can be redeclared and reassigned

```

//redeclaration and reassignment
var firstName="mehreen"
console.log(firstName)
var firstName="hira"
console.log(firstName)
//output
Mehreen
hira
```

Here we can see that we can reassign tha values in var.and we can redeclare variables in var

```
//redeclaration and reassignment
let firstName="mehreen"
console.log(firstName)
firstName="hira"
console.log(firstName)
//output
Mehreen
hira
let firstName="Gohar"
 console.log(firstName)
```

```
//output
SyntaxError: Identifier 'firstName' has already been declared
```

We can see here that let can be reassigned but it cannot be redeclared

```
const firstName="mehreen"
console.log(firstName)
 firstName="hira"
console.log(firstName)
//output
TypeError: Assignment to constant variable.

const firstName="mehreen"
console.log(firstName)
const firstName;
//output
SyntaxError: Identifier 'firstName' has already been declared
```

Here we can see that const variable can neither be re assigned nor redeclared

3)Hoisting:
Variables declared with var can be accessed before the line they are even declared.there places are occupied but the value stored in them is undefined when they are called before they are declared
For example

```
console.log(name)
var name="jhon"
console.log(name)
//output
Undefined
mehreen
```

Variables declared with let are hoisted differently.They are hoisted without a default initialization
Like

```
console.log(name)
let name="jhon"
//output
Cannot access 'name' before initialization

```

Similarly variables hoisting of const are likely in nature with let.
2)closures is basically nesting of functions and calling them dependently
Like

```
// //closure in javascript
 function addition(num1,num2){
     var sum=num1+num2
     function subtration(num){
         console.log(num-sum)
     }      return subtration
 }
 addition(3,4)(34)
//output
27
```

3)null: null refers to empty and it is a value in javascript.
Undefined:it is basically not recognized by the browser although the variable is not empty.the variable might have value but it is declared after it is called or might not have declared and only initialized
Like
Variable names=[ ]
This is null

```
function func1(id){
    console.log (id) }
func1()
```

This is undefined here as argument is empty so it will return undefined
4)this keyword refers to the object in which it is being initiated
Like class Names(name)={
constructor(
this.name=name)}
Here the name which comes through the parameter will be the name of the object
3)
1)the new features introduced in ES6 are
We can declare functions in a variable
Like

```
let name=function(firstname){
    return `the first name is ${firstname}`
}
console.log(name("mehreen"))
```

2)arrow function is the other form of a simple function

```
//normal function
function name(name){
    return name
}
console.log(name("mehreen"))
//arrow function
fname=()=>{
    return 'it returns the first name'
}
```

```
console.log(fname())
```

The difference in arrow function and simple function is of syntax and when simple function is written so even before it is called the whole function occupies the memory in the key of its name and in the case of arrow function before it is called infront of its name undefined is allocated in the memory.

3)

```
//tempelate literals
function addition(x,y){
    console.log (`the sum of the numbers are ${x+y}`)
}
addition(3,4)
//traditional string concatenation
function name(fname,lname){
    console.log("full name is "+fname +" "+ lname)
}
name("mehreen","fatima")
```

4)

1)Due to callback hell it was getting difficult for the developers to code.it was really confusing while calling the functions if we want to perform the functions one after one so promises came inorder to reduce confusions and the readability also enhanced.

```
 function add(x,y){
     return new Promise ((resolve)=>{
         resolve(sum =x+y)
     })

 }
 function subtract(sum,num){
     return new Promise ((resolve,reject)=>{
        resolve(result=sum-num)
     })
 }
 add(4,5)
 .then(function(res){
     console.log(res)
     return subtract(res,56)
     .then(function(res){
         console.log(res)
     })
 })
```

2)destructuring is basically restructuring of arrays or objects for our convenice when calling them like

```
//traditional method
let array1=["mehreen","fatima","ali"]
console.log([array1[0],array1[1],array1[2]])
//destructuring method
let [name1,name2,name3]=["mehreen","fatima","ali"]
console.log([name1,name2,name3])
//for objects
//traditional method
const obj={name:"mehreen",age:34,detailsOfStudent:["designer","problem
solver"]}
console.log(obj.detailsOfStudent)
//destructuring method
const
{name,age,detailsOfStudent:det}={name:"mehreen",age:34,detailsOfStudent:["
designer","problem solver"]}
console.log(det)
```

5)

1)The prototype chain in javascript is a method in which we can connect two objects so that the methods and properties present in them can be utilized by the other object.When children inherit something from the parents it means that the particular thing is present in the parent as well as in the children so if we want one object to inherit the properties and methods from the other object we create a prototype chain.

2)class Animals{

```
    constructor(name){
        this.name=name
    }
    eat(){
       return`${this.name} can eat`
    }
}
class Bunny extends Animals{

    hop(){
        return `${this.name} can hop`
    }
}
class Hare extends Bunny{
    color(){
      return  `and its color is gray`
    }
}
const animal1=new Bunny("rabbit")
const animal2=new Hare("Hare")
console.log(animal1.eat()+" and "+ animal1.hop())
console.log(animal2.eat()+" and "+ animal2.hop() + " and
"+animal2.color())
```

By this we can create chains and connect objects with each other
6)
1)Express js is a package and a framework..we have to install it to use it.we can make web app through it conveniently.we can only use node modules for it but with express js it is more easy for us and more quick too.it provides various tools for handling routing,serving static files and encoding url paths and what not
2).middleware is a function that have three parameters.res, req and next.req and res are objects and next is a function which create a chain between the other middleware functions.we have to use the next parameter if we have multiple middle ware functions in our code.

```
//middle ware function

const app=require("express")

const logger=function(req,res,next){

    console.log("logged")

}

app.use(logger)

app.get('/',(res,res)=>{

    res.send('Helloe people')

}).listen(8000)
```

Middle ware acts as a middle man between req and res.
3)
This is index.js file

```
const express = require('express')
const app = express()
const port = 8000
app.set("view engine","ejs")
app.use(express.urlencoded({ extended: true }))
app.get('/login', function(req, res) {
  let username="Jerry"
    res.render('pages/about',{username})
```

```
            })
app.listen(port)
```

This is about.ejs file which will be in views

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h2>Hello <%= username %></h2>
</body>
</html>
```