CN5006 WEEK-3

**Index.js**

//defination of the function EmployeeInfo

function EmployeeInfo(name,Salary)

{

console.log("Welcome " + name + ", Your monthly Salary is "+ Salary);

}

console.log("This is my second programe");

var EmpName="Sakar";

var EmpSalary=10000000000;

//calling of the function EmployeeInfo

EmployeeInfo(EmpName, EmpSalary);

//code for second exercise starts from here

const EmpSkills=(skills)=>

{

console.log("Sakar Shrestha is expert in "+ skills);

}

EmpSkills("java");

const student= require('./StudentInfo');

const person = require('./Person');

// because getName is the function so we use ()

console.log("Student Name:" +student.getName());

console.log(student.Location());

console.log(student.dob);

// because dob is a variable so we do nt use ()

console.log(student.Studentgrade(55));

console.log("grade is "+student.Studentgrade(55) );

person1= new person("Sakar","20","u2866703@uel.ac.uk");

console.log("using Person Module",person1.getPersonInfo());

console.log("Programe ended");

//Exercise 4

os=require("os")

const util=require('util')

console.log("temporary directory"+ os.tmpdir() )

console.log("hostname: "+ os.hostname())

console.log("OS : " + os.platform() +"release:"+ os.release())

console.log("Uptime"+ (os.uptime())/3600 +" hours")

console.log("userInfo" + util.inspect(os.userInfo()))

console.log("Memory "+ os.totalmem()/1000000000 + "Giga byte")

console.log(" free: "+os.freemem()/1000000000 + "Giga byte")

console.log("CPU "+ util.inspect(os.cpus()))

console.log("Network"+ util.inspect(os.networkInterfaces()))

console.log("programe end")

**Studentinfo.js**

//code for the StudentInfo starts from here

const dateofBirth= "16/09/2005";

const getStudentName = () =>

{

return "Sakar Shrestha";

}

const getCampusName = () =>

{return ("UEL Campus ")

}

//exporting functions & variable outside the module

exports.getName=getStudentName;

exports.Location=getCampusName;

exports.dob=dateofBirth;

// How to export function with parameters

exports.Studentgrade=(marks)=>

{

if (marks>50 && marks <70) return ("B grade");

else

return ("A grade");

};

**Person.js**

class student {

constructor(name, age, email) {

this.name = name;

this.age = age;

this.email = email;

}

getPersonInfo() {

return{

Name: this.name,

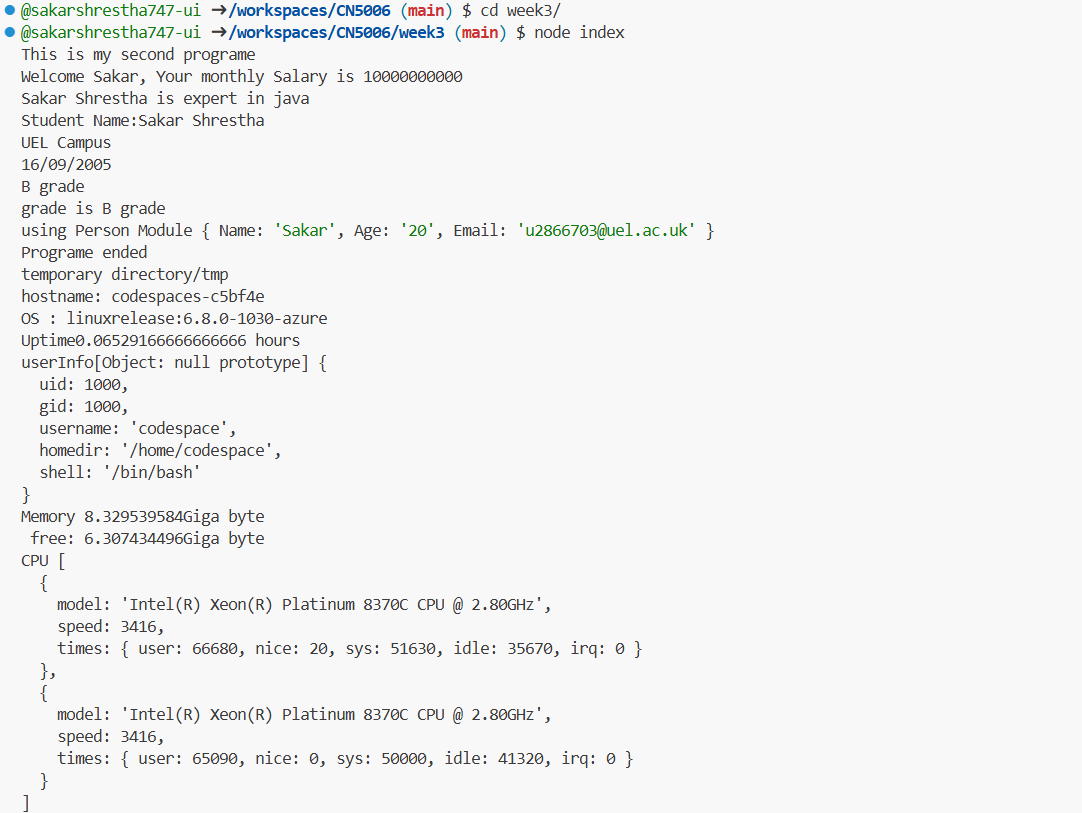
Age: this.age,

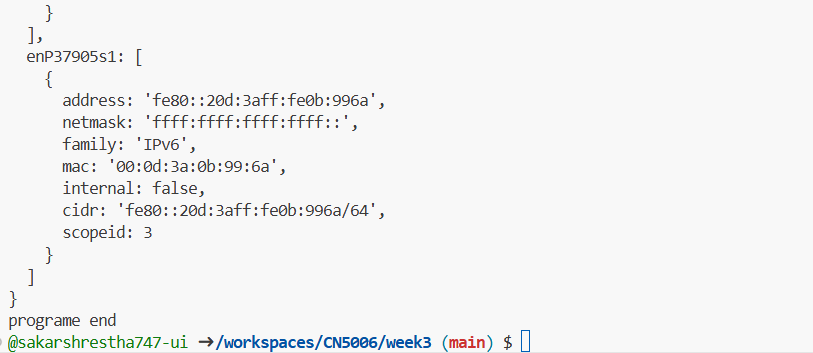
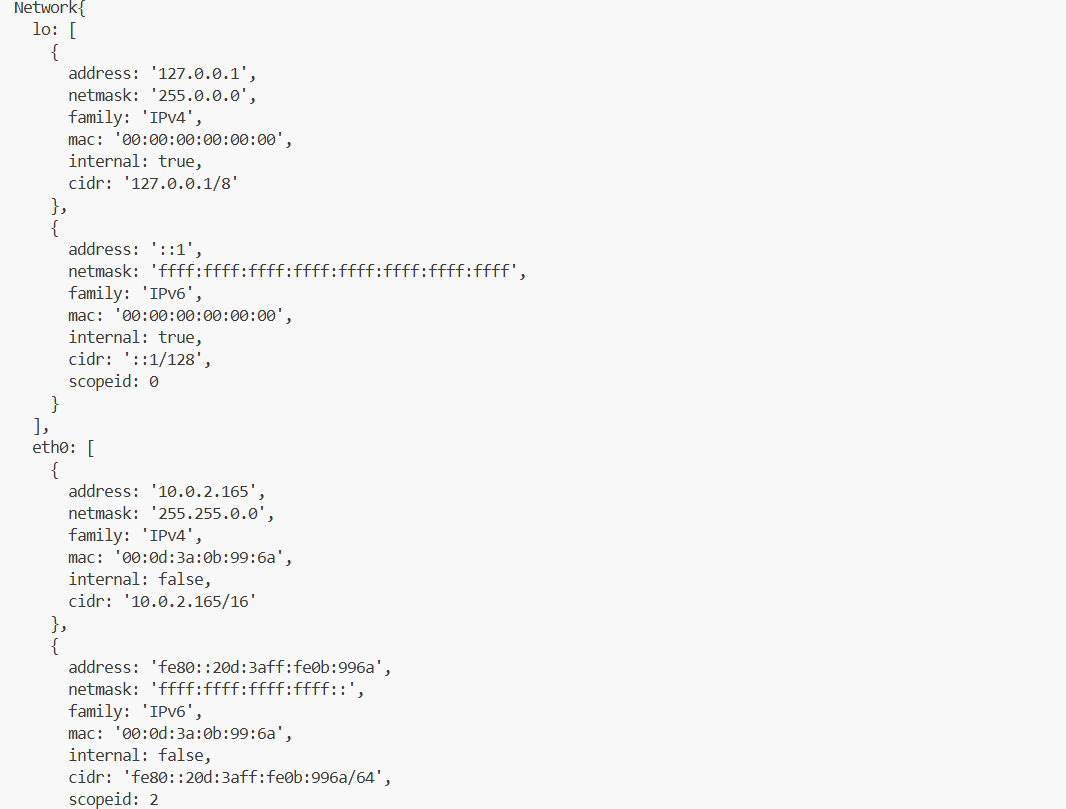
Email: this.email}

}

}

module.exports = student;





**Reflection of Week 3**

This hands-on, practical session gave participants a solid introduction to key JavaScript concepts, especially in the context of the Node.js runtime environment. The exercises significantly increased my confidence in writing structured, reusable code by logically progressing from basic programming structures to more complex strategies like modularity, object-oriented programming (OOP), and operating system (OS) interaction.  
  
Fundamentals, including defining and calling functions, were covered at the start of the session. A crucial contemporary JavaScript standard was illustrated by the swift transition from the conventional function EmployeeInfo() definition to the simple arrow function syntax. The significance of precise naming and parameterizing functions for logic encapsulation was emphasized by this first step.

I became aware of the differences between the two primary export techniques:

* Named Exports (exports. \*): Used in StudentInfo.js to export multiple items, including a logical function (Studentgrade), a variable (dob), and simple functions (getName, Location). This works well for utility files that contain several tools.
* Person.js uses the Default Export (module.exports) function to export the student class, which is the only primary entity.  
    
  These are being imported using Index's require () statement. Everything came together with js, demonstrating how dependencies are essential to the operation of a modular application.

**Putting Object-Oriented Principles into Practice**

Important object-oriented concepts were introduced through working with the Person.js file. Compared to traditional constructor functions, it felt much cleaner and more organized to define the blueprint for a student (which stood in for a person) using the ES6 class syntax. Data (name, age, email) and behavior can be clearly encapsulated in a single, reusable object structure using the constructor and the getPersonInfo() method. The use of new person(...) to instantiate this class in the main index file was a potent moment that linked the abstract idea of OOP to real-world implementation.

**Examining the Environment of Node.js**

Because it demonstrated Node.js's capabilities beyond standard web application logic, the final exercise, which made use of the built-in os and util modules, was especially instructive. Node.js is a robust system-level environment, as shown by the use of functions like os.tmpdir() and os.hostname() and the retrieval of comprehensive system metrics like memory (os.totalmem, os.freemem) and CPU information (os.cpus). The use of util.inspect() was crucial here for clearly displaying complex JavaScript objects like the userInfo and network interface data in a readable format.  
  
From basic function definition to intricate system introspection, this practical offered a strong basis, to sum up. I'm much more at ease using Node.js's built-in tools to manage application structure and access system resources, as well as structuring code into logical modules.