RIGHT STROKE FULL STACK TRAINING

Week-3 Assessment Typescript

1. Develop a program show casing the usage of Tuple data type. Assume you are getting a collection of Customer records and each element of customer records might have values of different data type. Represent the same as Collection of Tuple, iterate it, manipulate it and print it.

Program:

```
var customer1=["preetham",28,1200];

var customer2=["paul",23,1320];

var customer3=["karthik",8,12.50];

console.log("customer_name:"+customer1[0]+  "customer_age:"+customer1[1]+  "cu
stomer_totalpurchase:"+customer1[2])
console.log("customer_name:"+customer2[0]+  "customer_age:"+customer2[1]+  "cu
stomer_totalpurchase:"+customer2[2])
console.log("customer_name:"+customer3[0]+  "customer_age:"+customer3[1]+  "cu
stomer_totalpurchase:"+customer3[2])
customer1.push(15000);//push and pop
console.log(customer1)
customer2.push(13000);
console.log(customer2)
customer3.push(0);
console.log(customer3)
customer1.pop()
console.log(customer1)
var [c_name, c_age, c_purchase]=customer1;//destructing
console.log(c_name)
console.log(c_age)
console.log(c_purchase)
customer2[0]="abhi";//updating
console.log(customer2)
```

output:

```
"customer_name:preetham customer_age:28 customer_totalpurchase:1200"
"customer_name:paul customer_age:23 customer_totalpurchase:1320"
"customer_name:karthik customer_age:8 customer_totalpurchase:12.5"
["preetham", 28, 1200, 15000]
["paul", 23, 1320, 13000]
["karthik", 8, 12.5, 0]
["preetham", 28, 1200]
"preetham"
```

```
28
1200
["abhi", 23, 1320, 13000]
```

2. Develop a program that will calculate the surface area of a)Rectangle, b)Square, c)Triangle and make sure the precision with 2 decimal places.

a)Rectangle:

Sol:
```
var a=6;

var b=4.3;
var c=7.2;
var rec_prism=((2*a*b)+(2*b*c)+(2*c*a));
console.log("surface area of rectangular prism:"+rec_prism.toFixed(2))

output: "surface area of rectangular prism:199.92"
```

b)Square:

Sol:
```
var side=12.5;

var sq_cube=(6*side**2)
console.log("surface area of square cube:"+sq_cube.toFixed(2))

output: "surface area of square cube:937.50"
```

c)Triangle:

Sol:
```
var s1=12.5;

var s2=11.3;
var s3=10.45;
var h=3;
var b=5.2;
var l=9.21;
var tri_prism=(b*h+l*(s1+s2+s3));
console.log("surface area of triangular prism:"+tri_prism.toFixed(2))

output: "surface area of triangular prism:331.04"
```

3. Develop a program that exhibits inferred typing in Angular

Program:

```
var s1=12.5;
```

```
var s2=11.3;
var s3=10.45;
var h=3;
var b=5.2;
var l=9.21;
var tri_prism=(b*h+l*(s1+s2+s3));
console.log("surface area of triangular prism:"+tri_prism.toFixed(2))
tri_prism="nikitha";
console.log(tri_prism)
```

output: error: `var tri_prism: number`

```
Type '"nikitha"' is not assignable to type 'number'.
```

4. Develop a program that stores the days in a week in enum, and use the enum values in the program

Program:

```
enum Weekday {
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday,
    Sunday
}
namespace Weekday {
    export function isWorkingDay(day: Weekday) {
        switch (day) {
            case Weekday.Saturday:
            case Weekday.Sunday:
                return false;
            default:
                return true;
        }
    }
}

const mon = Weekday.Monday;
const sun = Weekday.Sunday;
console.log(Weekday.isWorkingDay(mon))
console.log(Weekday.isWorkingDay(sun))
```

output: [LOG]: true
[LOG]: false

5. Develop a program that exhibits union data type. And show case how an operations can be performed on these data stored in this variable. For example, store string and function data type on these variables

Program:

```
let in_Var : string | number;
   in_Var="Divya";
   console.log(in_Var);
   in_Var=34;
   console.log(in_Var);
   in_Var=true;
```

```
output [LOG]: "Divya"
[LOG]: 34
Type 'boolean' is not assignable to type 'string | number'.
```

```
      function display(value: (number | string))
      {
         if(typeof(value) === "number")
            console.log('The given value is of type number.');
         else if(typeof(value) === "string")
            console.log('The given value is of type string.');
      }
      display(true);
      display("ABC");
```

output: "The given value is of type string."

Error: Argument of type 'boolean' is not assignable to parameter of type 'string | number'.

6. Develop a Car Class in Typescript that has following attributes :- Car Color,Engine Capacity,No OfCylinders. And Methods:- StartCar(), StopCar(), AccelerateCar(), OpenCarLock(), CloseCarLock(). And Develop a program that will create instance of this class, and able to call its method

Program:

```
class car{
  colour;
  Engine;
  Capacity;
  NoofCylinders;
  constructor(colour:String, Engine:String, Capacity:Number, NoofCylinders:
number){
       this.colour=colour;
```

```
        this.Engine=Engine;
        this.Capacity=Capacity;
        this.NoofCylinders=NoofCylinders;
    }
    StartCar=()=>{
        return "started";
    }
    StopCar=()=>{
        return "stopped";
    }
    Accelarate=()=>{s
        return "accelerating";
    }
    OpenCarLock=()=>{
        return "opened";
    }
    CloseCarLock=()=>{
        return "closed";
    }
}
let Car:car = new car("magenta","XYZMotor",45,2);
console.log(Car.OpenCarLock())
console.log(Car.CloseCarLock());
console.log(Car.StartCar());
console.log(Car.Accelarate());
console.log(Car.StopCar());
console.log(Car);
```

output: [LOG]: "opened"
[LOG]: "closed"
[LOG]: "started"
[LOG]: "accelerating"
[LOG]: "stopped"
[LOG]: car: { "colour": "magenta", "Engine": "XYZMotor", "Capacity": 45, "Noofcylinders": 2 }

7. Covert the above Car class into Abstract class, and create following child classes :- SUV,HatchBack,Sedan. And create abstract methods in the base Car class, such as :- StartCar(),StopCar(). And create specific methods and behaviors in the related child classes.

Program:

```
 abstract class car{
  colour;
  Engine;
```

```
    Capacity;
    NoofCylinders;
    constructor(colour:String, Engine:String, Capacity:Number, NoofCylinders:
  number){
        this.colour=colour;
        this.Engine=Engine;
        this.Capacity=Capacity;
        this.NoofCylinders=NoofCylinders;
    }
  abstract startcar(): string;
  abstract stopcar(): string;
    abstract accelerate(): string;
    abstract opencarlock(): string;
    abstract closecarlock():string;
  }
  class SUV extends car {
    carname;
    constructor(carname:String,car_colour:String,Engine:String,Capacity:Numbe
  r,Noofcylinders:number){
         super(car_colour,Engine,Capacity,Noofcylinders);
         this.carname=carname;
    }
    startcar():string{
        return `The ${this.carname} is started`;
    }
  stopcar(): string{
        return `The ${this.carname} is stopped`;
    }
 accelerate(): string{
        return `The ${this.carname} is accelerating`;
    }
    opencarlock():string{
        return`The ${this.carname} lock is opened`;
    }
    closecarlock():string{
        return`The ${this.carname} lock is closed`;
    }
  }
class Hatchback extends car {
    carname;
    constructor(carname:String,car_colour:String,Engine:String,Capacity:Numbe
  r,Noofcylinders:number){
         super(car_colour,Engine,Capacity,Noofcylinders);
         this.carname=carname;
    }
    startcar():string{
        return `The ${this.carname} is started`;
    }
```

```
    stopcar(): string{
        return `The ${this.carname} is stopped`;
     }
     accelerate(): string{
        return `The ${this.carname} is accelerating`;
     }
     opencarlock():string{
        return`The ${this.carname} lock is opened`;
     }
     closecarlock():string{
        return`The ${this.carname} lock is closed`;
     }
   }
class Sedan extends car {
  carname;
     constructor(carname:String,car_colour:String,Engine:String,Capacity:Numbe
   r,Noofcylinders:number){
           super(car_colour,Engine,Capacity,Noofcylinders);
           this.carname=carname;
     }
  startcar():string{
        return `The ${this.carname} is started`;
     }
stopcar(): string{
      return `The ${this.carname} is stopped`;
     }
     accelerate(): string{
        return `The ${this.carname} is accelerating`;
     }
     opencarlock():string{
        return`The ${this.carname} lock is opened`;
     }
     closecarlock():string{
        return`The ${this.carname} lock is closed`;
     }
   }
   let Car:car = new SUV("suv","blue","pqr",25,3);
   console.log(Car.startcar());
   console.log(Car.stopcar());
   console.log(Car.accelerate());
   console.log(Car.opencarlock());
   console.log(Car.closecarlock());
   let Car1:car = new Sedan("sedan","blue","xyz",31,4);
   console.log(Car1.startcar());
   console.log(Car1.stopcar());
   console.log(Car.accelerate());
   console.log(Car.opencarlock());
   console.log(Car.closecarlock());
```

```
let Car2:car = new Hatchback("Hatchback","blue","lmn",15,5);
console.log(Car2.startcar());
console.log(Car2.stopcar());
console.log(Car.accelerate());
console.log(Car.opencarlock());
console.log(Car.closecarlock());
```

output:
```
[LOG]: "The suv is started"
[LOG]: "The suv is stopped"
[LOG]: "The suv is accelerating"
[LOG]: "The suv lock is opened"
[LOG]: "The suv lock is closed"
[LOG]: "The sedan is started"
[LOG]: "The sedan is stopped"
[LOG]: "The suv is accelerating"
[LOG]: "The suv lock is opened"
[LOG]: "The suv lock is closed"
[LOG]: "The Hatchback is started"
[LOG]: "The Hatchback is stopped"
[LOG]: "The suv is accelerating"
[LOG]: "The suv lock is opened"
[LOG]: "The suv lock is closed"
```

8. Create a Interface Payment manager, that has following abstract methods:- 1) public string doPayment(paymentcreds:string), 2) public string getPaymentStatus(refNumber : string) . And create following 2 implemented classes :- a)UPIPaymentManagerImpl , b) CreditCardPaymentManagerImpl

Program:
```
interface Paymentmanager{
  doPayment:(String)=>String;
  getPaymentStatus:(String)=>String

}
class paymentmanager implements Paymentmanager{
  refnum:String;
Credits:String;

  doPayment(Credits:String){
      this.Credits=Credits;
      return `The ${this.Credits} is credited`
  }
  getPaymentStatus(refnum:String){
     this.refnum=refnum;
     return `The ${refnum}is the reference number`;
  }
}
```

```
let p:Paymentmanager=new paymentmanager();
console.log(p.doPayment("100000"));
console.log(p.getPaymentStatus("cgdvswgd7y87989"))
```

output: [LOG]: "The 100000 is credited"
[LOG]: "The cgdvswgd7y87989is the reference number"

9. Develop a program that exhibits duck typing in Typescript

   Program:

```
function ducktyping(a:number,b:number){
return a+b;
}
let result=ducktyping(345,123);
console.log(result);
result=ducktyping(1,"pooja");
console.log(result);
result=ducktyping(5667,true);
console.log(result);
```

   output:468

   error: Argument of type 'string' is not assignable to parameter of type 'number'.//line 7
   Argument of type 'boolean' is not assignable to parameter of type 'number'.//line 9

10. Develop a program that will exhibit functions as :- a) Function with default parameter, b)Function with optional parameter, c) Function with Rest Parameter

a) Program:

```
function defaultpara(F_name:String,L_name:String){

return F_name+" "+L_name;
}
let result=defaultpara("neelam");
console.log(result);
let result2=defaultpara("praneeth","khan");
console.log(result2)
let result3=defaultpara("prachi","prakhyath","bajpai");
console.log(result3)
```

output: [LOG]: "neelam undefined"
[LOG]: "praneeth khan"
[LOG]: "prachi prakhyath"

   errors: Expected 2 arguments, but got 1.

Expected 2 arguments, but got 3.

b) Program:

```
function optionalpara(F_name:String,L_name?:String){

return F_name+" "+L_name;
}
let result=optionalpara("neelam");
console.log(result);
let result2=optionalpara("praneeth","khan");
console.log(result2)
let result3=optionalpara("prachi","prakhyath","bajpai");
console.log(result3)
```

output: [LOG]: "neelam undefined"
[LOG]: "praneeth khan"
[LOG]: "prachi prakhyath"

errors: Expected 1-2 arguments, but got 3.

c) Program:

```
function rpara(F_name:String, ...L_name:any[]){

return F_name+" "+L_name;
}
let result=rpara("setha","geetha",11213,7674.80);
console.log(result)
```

output: "setha geetha,11213,7674.8"