Typescript Coding ANSWERS

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, manipuate it and print it.

```
let  records:[String,number,String,String][];
records=[["Sai Avinash",1234,"male","California"],["Harshad",2345,"male","Vegus"],["Mehtha",2030,"male","Chicago"]];
records.push(["Jonny",2345,"male","Losangels"]);
records.pop();
for(var index of records){
console.log("Customer name is "+index[0]+" with "+index[1] +" as phone number, living in  "+index[3]);
}
```

Output :

PS D:\Angular RSC\myapp\src\app> node records.js
Customer name is Sai Avinash with 1234 as phone number, living in California
Customer name is Harshad with 2345 as phone number, living in Vegus
Customer name is Mehtha with 2030 as phone number, living in Chicago

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.

```
    export class
Triangle{
     tri_area=(h:number,b:number)=>{
return 0.5*h*b;
     }
  }  export class Square{
private a:number;
constructor(a:number){
this.a=a;
     }
     squa_area=()=>{
         return this.a*this.a;
     }
  }  export class
Rectangle{




    private len: number;     private bre:number;
private hei:number;
constructor(len:number,bre:number,hei:number){
this.len=len;          this.bre=bre;
this.hei=hei;
    }     rect_area=()=>{          console.log("The surface area of a rectangle is
"+2*((this.len*this.bre)+(this.br e*this.hei)+(this.hei*this.len)))
    } } let rectangle:Rectangle = new Rectangle(3,4,3);
rectangle.rect_area(); let square:Square = new Square(3);
console.log("The surface area of the square is "+square.squa_area());
let triangle:Triangle =new Triangle(); console.log("The Surface area of
the triangle "+triangle.tri_area(2,2));
```

3. Develop a program that exhibits inferred typing in Angular

```
function sum(a: number, b: number )
{     return a +
b;
} var total: number = sum(10,20); var str: string =
sum(10,20); //returns compilation error
console.log(total); console.log(str);
```

OUTPUT :

PS D:\Angular RSC\myapp\src\app> node records.js

30

PS D:\Angular RSC\myapp\src\app> tsc records.ts records.ts(6,5): error

TS2322: Type 'number' is not assignable to type 'string'.

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

```
enum Weekdays {
  Monday,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday,
  Sunday } console.log(Weekdays[1]); const getDay = () => 3;
console.log(Weekdays[getDay()]);
```

OUTPUT:

PS D:\Typescript> tsc enum.ts

PS D:\Typescript> node enum.js

Tuesday

Thursday

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

```
let val:(string | number)
val="sai";
console.log(val);
val=1234;
console.log(val);
//val=false; //return Compilation Error
function display(val: (string | number))
{     if(typeof(val) === "number")
console.log('Value is number.')
else if(typeof(val) === "string")
console.log('Value is string.')
}
display(123);
display("ABC");
```

OUTPUT:

PS D:\Typescript> tsc union.ts PS

D:\Typescript> node union.js

sai

1234

Value is number.

Value is string.

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 methods.

```
class Car {     color: string;     engine_capacity: number;
no_of_cylinders: number;     constructor(color: string, engine:
number,cylinders : number) {         this.color = color;
this.engine_capacity = engine;
this.no_of_cylinders=cylinders;
}     display():void {         console.log("COLOUR = " + this.color +  ",
Engine_capacity= " + this.engine_capac ity+ " NO_OF_CYLINDERS ="+ this.no_of_cylinders);
    }
    StartCar():void{
console.log("CAR STARTED....");
    }
    StopCar():void{
console.log("CAR STOPPED....");
    }
    AccelerateCar():void
{
    console.log("ACCELERATOR RAISED");
}
OpenLock():void{
    console.log("THE CAR LOCK IS OPENED");
}
CloseCarLock():void{     console.log("THE
CAR LOCK IS CLOSED");
} }
    let c=new Car('BLACK',2000,5);
c.display();









    c.StartCar();
    c.CloseCarLock();
```

OUTPUT:

PS D:\Typescript> tsc car.ts

PS D:\Typescript> node car.js

COLOUR = BLACK, Engine_capacity= 2000 NO_OF_CYLINDERS =5

CAR STARTED....

THE CAR LOCK IS CLOSED

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.

```
abstract class car{     private car_color;     private Engine;     private  Capacity;
private                                                              Noofcylinders;
constructor(car_color:String,Engine:String,Capacity:Number,Noofcylinders:number){
this.car_color=car_color;         this.Engine=Engine;         this.Capacity=Capacity;
         this.Noofcylinders=Noofcylinders;
    }
    Accelarate=()=>{         return `The car is accelerating
through ${this.Engine}`;     }
    OpenCarLock=()=>{
        return "The car door is opened";
    }
    CloseCarLock=()=>{
        return "The car door is closed";
    }     abstract startcar():
string;

    abstract stopcar(): string;
}
class shv extends car {
private carname:String;

constructor(carname:String,car_color:String,Engine:String,Capacity:Number,Noofcylinde
rs:number){
```

```
          super(car_color,Engine,Capacity,Noofcylinders);
this.carname=carname;
    }     startcar():string{         return
`The ${this.carname} is started`;
    }       stopcar(): string{         return
`The ${this.carname} is stopped`;
    }
 } class sedan extends car{     private carname:String;
constructor(carname:String,car_color:String,Engine:String,Capacity:Number,Noofcylinde
rs:number){           super(car_color,Engine,Capacity,Noofcylinders);
this.carname=carname;
    }     startcar():string{         return
`The ${this.carname} is started`;
    }       stopcar(): string{         return
`The ${this.carname} is stopped`;
    } } class Hatchback extends car{     private carname:String;
constructor(carname:String,car_color:String,Engine:String,Capacity:Number,Noofcylinde
rs:number){           super(car_color,Engine,Capacity,Noofcylinders);
this.carname=carname;
    }
    startcar():string{
        return `The ${this.carname} is started`;
    }
    stopcar(): string{         return `The
${this.carname} is stopped`;
    }
}   let emp:car = new shv("shv","Grey","nd-
23",23,34); console.log(emp.OpenCarLock());
console.log(emp.CloseCarLock());
console.log(emp.startcar());
console.log(emp.Accelarate());
console.log(emp.stopcar()); console.log('--------
----------------');
```

```
let emp1:car = new sedan("sedan","white","Ng-23",29,39);
console.log(emp1.OpenCarLock()); console.log(emp1.CloseCarLock());
console.log(emp1.startcar()); console.log(emp1.Accelarate());
console.log(emp1.stopcar()); console.log('------------------------'); let
emp3:car = new Hatchback("Hatchback","Black","M-23",13,24);
console.log(emp3.CloseCarLock()); console.log(emp3.OpenCarLock());
console.log(emp3.startcar()); console.log(emp3.Accelarate());
console.log(emp3.stopcar());
```

OUTPUT :

PS D:\Typescript> The car door is opened

>> The car door is closed

>> The shv is started

>> The car is accelerating through nd-23

>> The shv is stopped

>> -------------------------

>> The car door is opened

>> The car door is closed

>> The sedan is started

>> The car is accelerating through Ng-23

>> The sedan is stopped

>> ------------------------ >>

The car door is closed

>> The car door is opened

>> The Hatchback is started

>> The car is accelerating through M-23

>> The Hatchback is stopped

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

```
9.      interface Payment{
10.     doPayment:(String)=>String; 11.
getPaymentStatus:(String)=>String
12.
13.      }
14.      class Bank implements Payment{
15.      refnum:String;
16.      paymentCredits:String;
17.
18.     doPayment(paymentCredits:String){
19.     this.paymentCredits=paymentCredits;
20.     return `The ${this.paymentCredits} is credited to your account` 21.      }
22.         getPaymentStatus(refnum:String){
23.         this.refnum=refnum;
24.         return `The ${refnum} reference number payment is successfull`;
25.         }
26.         }
27.         let e:Payment=new Bank();
28.         console.log(e.doPayment("1,00,000"));
29.         console.log(e.getPaymentStatus("678945434345"))
```

OUPTUT:

PS D:\Typescript> node Abstract.js
The 1,00,000 is credited to your account
The 678945434345 reference number payment is successful

9. Develop a program that exhibits duck typing in Typescript

```
class A {
name="Sai Avinash";
}
class B
{
    name = "Mehtha";
}
let a: A = new A(); // substitutes   let
b: A = new B();

console.log("A name : "+a.name);
console.log("B name : "+b.name);
```

OUTPUT :

PS D:\Typescript> tsc duck.ts

PS D:\Typescript> node duck.js

A name : Sai Avinash

B name : Mehtha

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

```
//DEFAULT PARAMETES function Introduce(name: string, frame: string =
"Hello,I am ") : string {     return frame + ' ' + name + '!';
}  console.log(Introduce('Sai Avinash'));
console.log(Introduce('Sai Avinash', 'Good morning sir,This is
'));
//OPTIONAL PARAMATER function College(name: string,
college?: string ) : string {     return name + ' : ' +
college ;
}  console.log(College('Sai
Avinash','GIET')); console.log(College('Sai
Avinash'));

//REST PARAMETER function Team(start: string,
...names: string[]) {     return start + " " +
names.join(", ") + "!";
}
console.log(Team("Hello", "Abhishek","Harish","Sohal"));
console.log(Team("Hello"));
```

OUTPUT :

PS D:\Typescript> tsc func.ts PS

D:\Typescript> node func.js

Hello,I am Sai Avinash!

Good morning sir,This is Sai Avinash!

Sai Avinash : GIET

Sai Avinash : undefined Hello

Abhishek, Harish, Sohal!

Hello !