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.**

var customer1=["raj",28,1200];

var customer2=["neelam",23,1320];

var customer3=["karan",8,12.50];

console.log("customer\_name:"+customer1[0]+  "customer\_age:"+customer1[1]+  "customer\_totalpurchase:"+customer1[2])

console.log("customer\_name:"+customer2[0]+  "customer\_age:"+customer2[1]+  "customer\_totalpurchase:"+customer2[2])

console.log("customer\_name:"+customer3[0]+  "customer\_age:"+customer3[1]+  "customer\_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]="prashant";//updating

console.log(customer2)

**output:**

"customer\_name:raj customer\_age:28 customer\_totalpurchase:1200"   
"customer\_name:Neelam customer\_age:23 customer\_totalpurchase:1320"   
"customer\_name:karan customer\_age:8 customer\_totalpurchase:12.5"   
["raj", 28, 1200, 15000]   
["neelam", 23, 1320, 13000]   
["karan", 8, 12.5, 0]   
["raj", 28, 1200]   
"raj"   
28   
1200   
["prashant", 23, 1320, 13000]

1. **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) 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) 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) 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**

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'.

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

1. **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 in\_Var : string | number;

in\_Var="poojitha";

console.log(in\_Var);

in\_Var=34;

console.log(in\_Var);

in\_Var=true;

**output:** [LOG]: "poojitha"   
[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'.

1. **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**

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 }

1. **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{

  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:Number,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:Number,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:Number,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"

1. **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**

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"

1. **Develop a program that exhibits duck typing in Typescript**

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

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

a) 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) 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) 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"