**PART – 01**

**Introduction**

**(we need to declare datatype for everything)**

**PART – 02**

**Installation:**

1. Install node js:

Open below link:

[**https://nodejs.org/en/**](https://nodejs.org/en/)

and install it. To check node js is install or not , hit below command on command prompt:

**node -v**

1. Install vs code editor by using below link:

<https://code.visualstudio.com/download>

1. Install type script:

Create a project folder (with any name) than go to inside project folder by vs code using terminal (ctrl + `) and than hit below command:

**npm install -g typescript**

To check that typescript is installed or not, hit below command inside project folder with in vs code:

**tsc --v**

Suppose that we have a folder name of **TS ,** then open this folder with in vs code:

Create a new **index.ts** inside TS folder and paste below code:

console.log("welcome to typescript");

let num = 5;

/\*  where number is datatype or type annotation, if we will not declare datatype then it will automatically consiter number according to value \*/

num = 55;

than we need to compile than we need to run .

1. To compile:

Run below command in terminal of vs code inside project folder (TS):

**tsc** index.ts

Note:

1. It will generate index.js file automatically.
2. if we will use only tsc command then it will compile all files.
3. To Run:

Run below command in terminal of vs code inside project folder(TS):

**node** index.js

1. We need to use js file in our html file. Like :

<script src=”./index.js” type=””module> </script>

**PART – 03**

**(Deep dive , errors, solving and ts config)**

**Ts configuration file:**

Run below command inside project folder(TS):

**tsc --init**

Note:

1. It will generate tsconfig.json file.
2. Search **noEmitOnError** and enable it.
3. If ts file have an any error than tsconfig.json file will prevent to generate js file and we have to use below command to compile:

**tsc**

**OR**

We can Run below command to compile:

**tsc -- noEmitOnError** index.ts

**Addition of two number:**

Open index.ts file and replace below code:

function sum(a:number,b:number):number //this is parameter

{

    return (a+b);

}

/\*          OR

const sum = (a:number,b:number):number =>

{

    return (a+b);

}

\*/

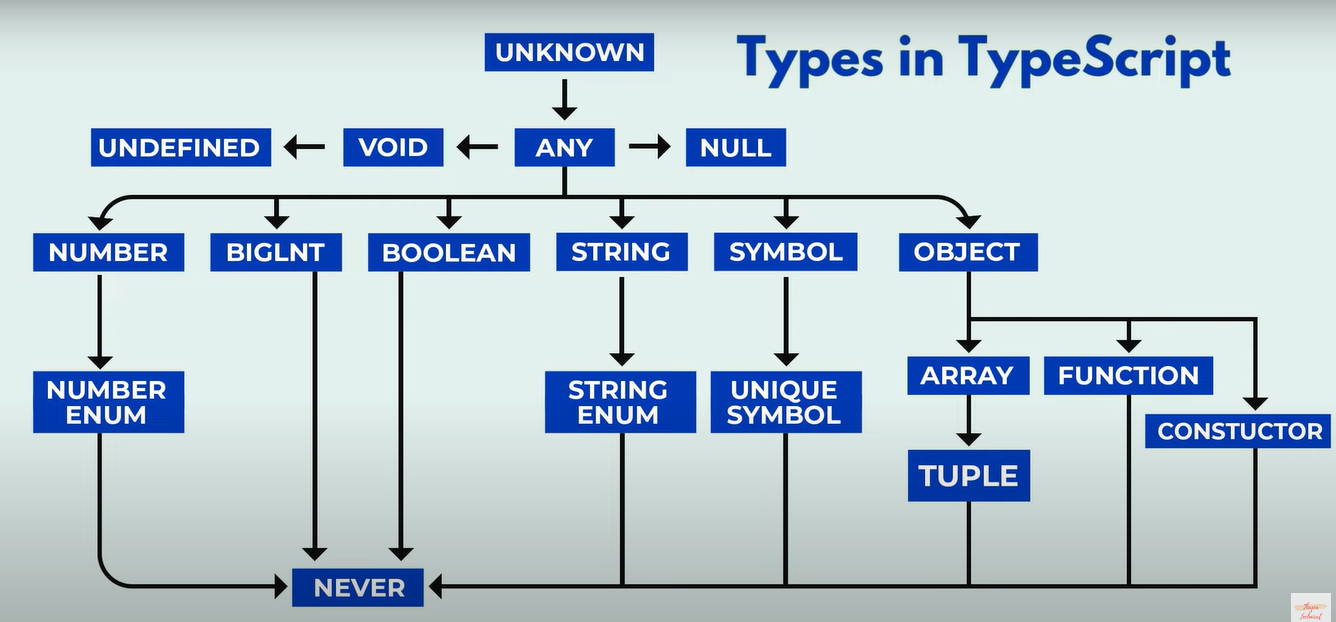
console.log(sum(5,10)); //this is arguments

than compile than run.

Note: By default return type of function is void.

**PART – 04**

**(type annotation)**



**PART – 05**

**(Boolean and bigint)**

**Boolean**

Open index.ts file and replace below code:

const sum = (a:number):boolean =>

{

    return a%2===0;

}

console.log(sum(5));

than compile than run.

**Bigint**

Open index.ts file and replace below code:

let bignumber:bigint = 9007199254740992n;

/\*                OR

let bignumber:bigint = BigInt("90071992547409925");

\*/

console.log(bignumber);

Note:

1. open tsconfig.json file than search **target** and reset its value with “**es2022”**
2. open tsconfig.json file than enable **lib** and put its value as “**es2022” , “DOM”**

like:

"lib": ["es2022","DOM"],

than compile than run.

**PART – 06**

**(any and unknown)**

**any**

We can store any type data.

**unknown**

still unknown but in future we can store any type and according to type of data we can perform any expression on condition basis.

let num:unknown;

num = 5;

num = "hello";

num = true;

if(typeof num === "number")

{

    console.log(num+5);

}

else if(typeof num === "boolean")

{

    console.log(num);

}

than compile than run.

**PART – 07**

**(function invocation or calling, declaration and return type)**

Open index.ts file and replace below code:

const greet = (name:string , id:number):string=>{

    return `welcom ${name} and your id is ${id}`;

}

console.log(greet("Rakesh",50));

than compile than run.

It’s return type is string and bydefault function’s return type is void.

**PART – 08**

**(type inference)**

Ex:

let sum:number = 10;

Note:

1. here we have declared type is number, i.e. we can store interger type data only. If we we will not declare than type script compiler automatically consider it’s type according to value, that is called type inference.

**PART – 09**

**(default and optional parameter)**

**Default parameter:**

Open index.ts file and replace below code:

const greet = (name:string , id:number = 1):string=>{

    return `welcom ${name} and your id is ${id}`;

}

console.log(greet("Rakesh"));

than compile than run.

Note:

1. we should be pass two arguments but we have passed one argument and id is default parameter with value 1

**Optional parameter:**

Open index.ts file and replace below code:

const greet = (name:string , id?:number):string=>{

    if(id)

    {

        return `welcome ${name} and id is ${id}`;

    }

    else

    {

        return `welcome ${name}`;

    }

    return `welcome ${name}`;

}

console.log(greet("Rakesh"));

than compile than run.

Note:

1. we should be pass two arguments but we have passed one argument and id is optional parameter with the keyword ?

**PART – 10**

**(Array)**

There are three way to define array:

1. **using square brackets**:

let numbers:number[] = [1,2,3,4,5];     //we can store number of array

1. **using array constructor**:

let numbers2:number[] = new Array(1,2,3,4,5);     //we can store number of array

1. **using array of method**:

let numbers3:string[] = Array.of("hello","welcome");     //we can store string of array

**PART – 11**

**(Array operation and iteration)**

Same as java script

**PART – 12**

**(map and filter function)**

Open index.ts file and replace below code:

const numbers:number[] = [1,2,3,4,5];

const doubleData:number[] = numbers.map((curVal:number)=>curVal\*2);

console.log(doubleData);

const numberToString:string[] = numbers.map((curElm:number)=>curElm.toString());

console.log(numberToString);

const evenNumbers:number[] = numbers.filter((curElm:number)=> curElm %2 === 0);

console.log(evenNumbers);

const nogreaterthanthree = numbers.filter((curElem:number)=> curElem > 3);

console.log(nogreaterthanthree);

than compile than run.

**PART – 13**

**(object)**

Open index.ts file and replace below code:

const person:{name:string;age:number;isStudent:boolean;address:{city:string;country:string}} =

{

    name:'Deepanshu',

    age:27,

    isStudent:true,

    address:{ city:"pune", country:"india" }

};

console.log(person);

than compile than run.

**PART – 14**

**(Type Alies)**

Open index.ts file and replace below code:

1st example:

type Person = {  //first letter must be capital,  this is type alias

    name:string;

    age:number;

    isStudent:boolean;

    address:{city:string;country:string};

    class?:string;     //optional property

};

const person:Person =

{

    name:'Deepanshu',

    age:27,

    isStudent:true,

    address:{ city:"pune", country:"india" },

    class:'seventh'

};

console.log(person);

const person2:Person =

{

    name:'Deepanshu2',

    age:272,

    isStudent:true,

    address:{ city:"pune2", country:"india2" }

};

console.log(person2);

than compile than run.

Open index.ts file and replace below code:

2nd example:

type Product = {  //first letter must be capital,  this is type alias

    name:string;

    price:number;

    quantity:number;

};

const product:Product =

{

    name:'Laptop',

    price:1000,

    quantity:5

};

const calculateTotalPrice = (product:Product)=>{

    /\* return product.price \* product.quantity;

                OR \*/

    return `${product.name} total cost ${product.price \* product.quantity}`;

};

console.log(calculateTotalPrice(product));   //we have passed object

than compile than run.

**PART – 15**

**(Call Signature)**

Open index.ts file and replace below code:

type Student = {  //first letter must be capital,  this is type alias

    name:string;

    age:number;

    gender?:string;

    greet:(country:string)=>string;      //method call signature

                                    // it will take an parameter and return an string value

};

const student1:Student =

{

    name:'Vinod',

    age:29,

    greet:(country):string => `Welcome my name is ${student1.name}, I am ${student1.age} year old and i am from ${country}`

};

const student2:Student =

{

    name:'Thapa',

    age:39,

    greet:(country):string => `Welcome my name is ${student2.name}, I am ${student2.age} year old and i am from ${country}`

};

const introduction = (student1:Student) =>{

    const{name,age} = student1;

    return `Welcome my name is ${name}, I am ${age} year old`;

};

console.log(introduction(student1));

console.log(student1.greet('india'));

console.log(student2.greet('america'));

than compile than run.

**PART – 16**

**(Enum)**

One property can have multiple values.

Open index.ts file and replace below code:

enum Roles{user="user",admin="admin"}

type LoginDetails = {  //first letter must be capital,  this is type alias

    name?:string;

    email:string;

    password:string;

    role:Roles;

  };

  const user1:LoginDetails = {

    name:"vinod",

    email:"xyz@gmail.com",

    password:"cudnsj",

    role:Roles.admin

  };

  const user2:LoginDetails = {

    email:"abc@gmail.com",

    password:"cnnd78",

    role:Roles.user

  };

  const isAdmin = (user:LoginDetails) =>{

    const{name,role,email} = user;

    return role === "admin" ?  `${name} this is admin`  :  `${email} this is user`

  };

  console.log(isAdmin(user1));

  console.log(isAdmin(user2));

than compile than run.

**PART – 17**

**(Tuples)**

It is similar to array.

Open index.ts file and replace below code:

type PersonInfo = readonly [string,number,boolean];  //first letter must be capital letter

                    //we can not push/edit any value

const displayPersonInfo = (person:PersonInfo) =>{

    const[name,age,hasDrivingLiecence] = person;

    console.log(`Name is ${name}, Age is ${age}, Liecense is ${hasDrivingLiecence ? "Yes" : "No"}`);

};

const person1:PersonInfo = ['vinod',29,true];

const person2:PersonInfo = ['vinod2',229,false];

//const person2:PersonInfo = [29,'vinod',true];  //it will get error due to order mismatch

displayPersonInfo(person1);

displayPersonInfo(person2);

than compile than run.

**PART – 18**

**(Unions and intersection)**

Union:

Open index.ts file and replace below code:

const inputValue = (value:string | number | boolean)=>{  //can receive three types data

    if(typeof value === 'string')

    {

        return value.toUpperCase();

    }

    else if(typeof value === 'number')

    {

        return value \* 2;

    }

    else

    {

        return('invalud input data');

    }

};

console.log(inputValue(2000));

console.log(inputValue('welcome'));

console.log(inputValue(true));

than compile than run.

Intersection:

Open index.ts file and replace below code:

type Person = {         // first letter must be capital letter

    name:string;

    age:number

};

type Employee = {

    emp\_id:number;

    department:string

};

type EmployeeIntersection = Person & Employee            // we need to use both type

const exp1:EmployeeIntersection = {                      //this is intersection

    name:"Rahul",age:34,emp\_id:20290,department:"software"

};

console.log(exp1);

type EmployeeUnion = Person | Employee     // we can use only one type

const exp2:EmployeeUnion =  {              //this is union

    name:"Rakesh" , age: 45

};

console.log(exp2);

const createProfile = (myexp1:EmployeeIntersection,myexp2:EmployeeUnion)=>{

    return {...myexp1 , ...myexp2};

};

const completeinfo = createProfile(exp1,exp2);

console.log(completeinfo);

than compile than run.

**PART – 19**

**(Generics)**

We can send and receive any type data to function.

Open index.ts file and replace below code:

function logAndReturn<Thapa>(value:Thapa):Thapa{   //we can receive any type data

    return value;

}

const numberResult:number = logAndReturn<number>(42);   //we can send any type data

const stringResult:string = logAndReturn<string>("Hello Generics");   //we can send any type data

const booleanResult:boolean = logAndReturn<boolean>(true);  //we can send any type data

console.log(numberResult);

console.log(stringResult);

console.log(booleanResult);

**OR**

function logAndReturn<Thapa>(value:Thapa):Thapa{   //we can receive any type data

    return value;

}

const numberResult = logAndReturn(42);   //we can send any type data

const stringResult = logAndReturn("Hello Generics");   //we can send any type data

const booleanResult = logAndReturn(true);  //we can send any type data

console.log(numberResult);

console.log(stringResult);

console.log(booleanResult);

than compile than run.

**PART – 20**

**(inlay hints)**

Open vs code editor, press **ctrl + ,** than search **typescript inlay hints** and enable all things.

Than we can get hints during coding.

**PART – 21**

**(function overloading with generics)**

Open index.ts file and replace below code:

function add<T,U>(a:T,b:U):void{   //we can receive any type data

    console.log(typeof a);

    console.log(typeof b);

}

const result1 = add<number,string>(5,"thapa");

const result2 = add("thapa",5);

than compile than run.

**PART – 22**

**(interface)**

Open index.ts file and replace below code: (need to use predefined keyword interface)

interface Products{      //same as type alias

    name:string;

    price:number;

    quantity:number;

}

const product1:Products={

    name:"vinod",

    price:10000,

    quantity:5

}

const product2:Products={

    name:"Laptop",

    price:200000,

    quantity:50

}

const calculateTotalPrice = (product:Products):number =>{

    const{price,quantity} = product;

    return price \* quantity ;

};

console.log(calculateTotalPrice(product1));

console.log(calculateTotalPrice(product2));

than compile than run.

**PART – 23**

**(compiler)**

Generally after editing we need to compile our ts file. By using watch mode we don’t require to compile frequently.

It will automatically compile.

We need to hit below 2 command:

(1) **tsc --init**

(2) **tsc --watch**  //for all files

**OR**

**tsc** index.ts **--watch** for particular file

**Note:** we can use **--watch** or **-w**

**////////////////////////////////////////////////////////////////////**

To generate js files in a separate folder:

1. open tsconfig.json file and search **rootDir** and enable it. And replace it with below code:

"rootDir": "./src",

1. Again search **outDir** and enable it. And replace it with below code:

 "outDir": "./dist",

Note:

1. **src** is our root folder (we can put any name) where all ts file will save.
2. **dist** is our another created folder where all js file will generate.
3. Both folders location is same.
4. We need to use js file in our html file. Like :

<script src=”./index.js” type=””module> </script>

**PART – 24**

**(creating typescript website)**

Create a project folder suppose **TS** and open this folder with in vs code editor. Create a new file **index.html** inside project folder. Create a folder **css** inside project folder. Create a file **style.css** inside css folder. Create 2 folders **src** and **dist** inside project folder. Create a file **index.ts** inside src folder.

Than open the terminal and run below command to generate config file:

**tsc --init**

open the config file and search **target** and update it with **es2022** instead of es2016

than search **rootDir** and update it with **“./src”,** instead of “./”,

than search **outDir** and enable it, and update it with **“./dist”,** instead of “./”,

than search **noEmitOnError** and enable it. (if having error than it will not generate corresponding js file)

Open index.ts file and add below code :



Than run below command on terminal:

**tsc**

Note: it will generate corresponding index.js file inside dist folder.

Than we will attach the js file inside our index.html file with in body or head tag:



Now we can run the website by npm start or by installing live server

Note: we need to write the code inside index.ts file it will generate js code automatically.

**PART – 25**

**(OOPS concept, class and constructor)**

We need to use predefined keyword **class**

Open index.ts file and replace below code

Example 1:

class Persons{

    name:string = "Rahul";

    age:number = 55;

    hobbies:string[] = ["Reading","Painting"];

}

const person1:Persons = new Persons();

console.log(person1);

than compile than run.

Example 2:

class Persons{

    name:string;

    age:number;

    hobbies:string[];

    constructor(name:string,age:number,hobbies:string[])

    {

        this.name=name;

        this.age=age;

        this.hobbies=hobbies;

    }

}

const person1:Persons = new Persons("Rahul",55,["Reading","Painting"]);

console.log(person1);

than compile than run.

**PART – 26**

**(Inheritance)**

Open index.ts file and replace below code

class Persons{

    name:string;

    age:number;

    hobbies:string[];

    constructor(name:string,age:number,hobbies:string[])

    {

        this.name=name;

        this.age=age;

        this.hobbies=hobbies;

    }

    introduce():string

    {

        return `Hi i am ${this.name} and i am ${this.age} year old and i love ${this.hobbies.join(",")}`

    }

}

class Students extends Persons{

    grade:number;

    constructor(grade:number)

    {

        super("Rahul",55,["Reading","Painting"]);

        this.grade = grade;

    }

}

const person1:Persons = new Persons("Rahul",55,["Reading","Painting"]);

console.log(person1);

console.log(person1.introduce());

const person2:Students = new Students(666);

console.log(person2);

console.log(person2.introduce());

than compile than run.

**PART – 27**

**(visibility mode: public,private and protected)**

**(visibility mode or access modifier)**

**public** , **private** and **protected**

Note: Default visibility mode is **public** . Concept same as C++, we need to use before variable or function declaration.

Ex:

public name:string;

    private age:number;

    protected hobbies:string[];

**PART – 28**

**(shorthand properties)**

Open index.ts file and replace below code

/\*

class Persons      //using simple methos

{

    name:string;

    age:number;

    hobbies:string[];

    constructor(name:string,age:number,hobbies:string[])

    {

        this.name = name;

        this.age=age;

        this.hobbies=hobbies;

    }

}

\*/

class Persons      //using short hand properties

{

    constructor(private name:string,private age:number,private hobbies:string[])

    {

    }

}

const person1:Persons = new Persons("Rahul",55,["Reading","Painting"]);

console.log(person1);

than compile than run.

**PART – 29**

**(getter and setter method)**

Open index.ts file and replace below code

class Persons

{

    private \_age:number | undefined;      //we will asign value

    constructor(public name:string, protected hobbies:string[])

    {

    }

    public set age(age:number)      // we use set for setter method, it takes only one parameter

    {

        if(age > 150 || age < 0)

        {

            throw new Error('Age is not valid');

        }

        this.\_age = age;

    }

    public get age()      // we use get for getter method, it doesn't take any parameter

    {

        if(this.\_age === undefined)

        {

            throw new Error('Age is not defined');

        }

        return this.\_age;

    }

    introduceParent():string

    {

        return `Hi i am ${this.name}, i am ${this.\_age} yr old and my hobbies is ${this.hobbies}`

    }

}

const person1:Persons = new Persons("Rahul",["Reading","Painting"]);

person1.age = 12;       // calling setter mthod

console.log(person1.introduceParent());

console.log(person1.age);      //calling getter method

than compile than run.

**PART – 30**

**(Practice)**

**(Bank account and temperature convertor)**

Practice 1:

Open index.ts file and replace below code

class BankAccount

{

    private \_balance:number = 0;

    public get balance():number

    {

        return this.\_balance;

    }

    public set balance(newBalance:number)

    {

        if(newBalance < 0)

        {

            throw new Error('Invalid Balance');

        }

        this.\_balance = newBalance;

    }

};

const account = new BankAccount();

account.balance = 10;

console.log(account.balance);

than compile than run.

Practice 2:

Open index.ts file and replace below code

class Temerature

{

    private \_celsious:number = 0;

    public get celsious():number

    {

        return this.\_celsious;

    }

    public set celsious(newCelsious:number)

    {

        this.\_celsious = newCelsious;

    }

    public get fehranhit():number

    {

        return(this.\_celsious \* 9) / 5 + 32;

    }

    public set fehranhit(newFah:number)

    {

        this.\_celsious = (newFah - 32) \* 5 / 9;

    }

};

const temp:any = new Temerature();

temp.celsious =25;

console.log(temp.fehranhit);

temp.fehranhit = 77;

console.log(temp.celsious);

than compile than run.

**PART – 31**

**(static variable and static method)**

Open index.ts file and replace below code

class MathOperations

{

    public static PI:number = Math.PI;

    public static add(num1:number , num2:number)

    {

        return (num1 + num2);

    }

};

console.log(MathOperations.PI);

console.log(MathOperations.add(20,50));

than compile than run.

Note: concept same as c++.

**PART – 32**

**(abstract class)**

Open index.ts file and replace below code

abstract class Shape

{

    constructor(protected color:string){}

    abstract calculateArea():number;

    abstract displayArea:()=> void;

};

class Circle extends Shape

{

    constructor(protected color:string , protected radius:number)

    {

        super(color);

    }

    public calculateArea():number

    {

        return(Math.PI)\*this.radius\*this.radius;

    }

    displayArea=()=>

    {

        console.log(`this is a ${this.color} color circle with radius ${this.radius}`);

    }

};

const circle = new Circle('red',5);

console.log(circle.calculateArea());

circle.displayArea();

than compile than run.

Note: concept same as java.

**PART – 33**

**(type v/s interface)**

**(watch another video)**

**PART – 34**

**(type safety)**

**(watch another video)**

**PART – 35**

**(Only suggestion to use functional component)**

**PART – 36**

**(working with PARCEL)**

**(steps to create a website using html,css,typescript and parcel(alternative of webpack))**

**Parcel:** Parcel bundles html,css and typescript code.

**Step (1) :**

Create a project folder suppose **PARCEL\_TS** . open this folder with in vs code editor than go to terminal and run below

command:

**npm init -y**

Again run below command for parcel:

**npm i parcel --save -dev**

**Step(2):**

Create another folder suppose **src** inside PARCEL\_TS folder, than create two new file **index.ts** and **index.html** inside src folder. Than create a new folder **css** inside project folder. Create a new **style.css** inside css folder.

**Step(3):**

Open index.html file and add below code inside body tag:

We can attach <script> tag inside <head> tag.



**Step(4):**

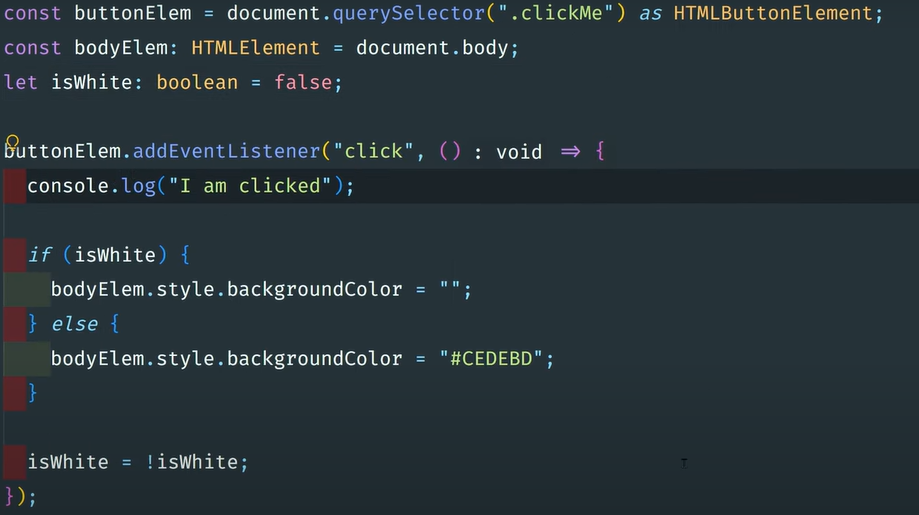
Go to terminal and run below command:

**tsc -init**

Note: it will generate config file.

Open config file and search **noEmitOnError** than enable and true it.

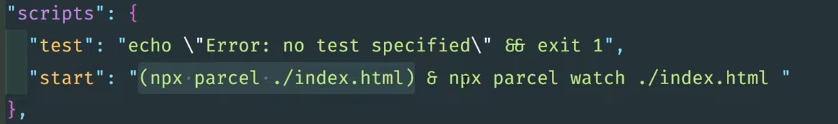
Open index.ts file and add below code:



Open package.json file and remove below code:



Than update the script with below code:



Than run below command in terminal to start development server:

**npm start**

Note: it will automatic compile.