1. **Typescript** is a superset of javascript , this statement here doesn't mean that it provides any kind of additional features than javascript but it simply means that it helps in writing JS in such a manner that we will face much less errors.

2. All the code that we write into typescript is finally compiled into javascript.

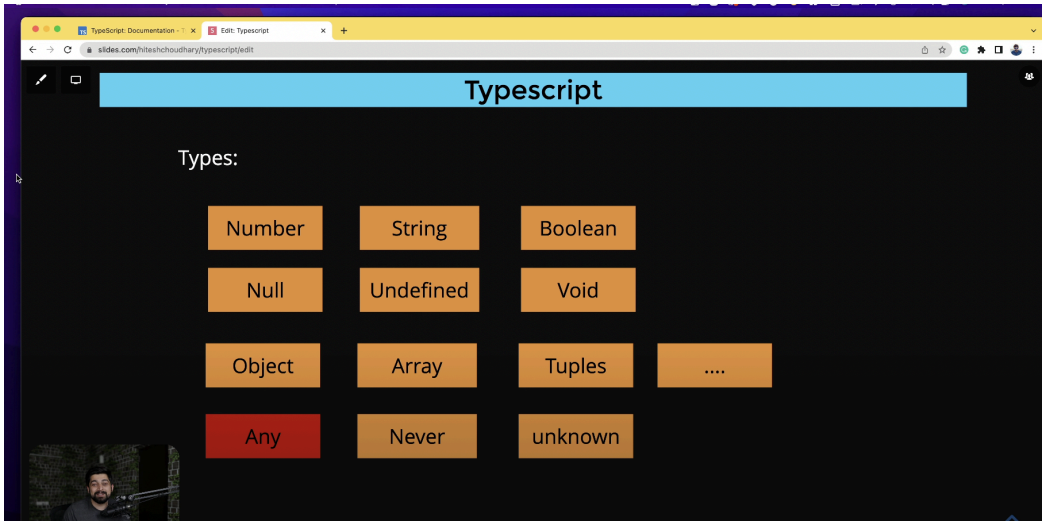
3. We can do many things in js which shouldn't be allowed for example we can add 2 + "2" In js which gives result as 22 , at first js must throw an error of adding a number with string but js is not much concerned about this but typescript is.

Typescript is not what you think

Typescript only does static checking i.e. it only analyzes the syntax of the written code which is not the case with the javascript.In javascript when the code is run in the node environment then only the errors will be thrown

```
tsc filename //this command will change .ts file to .js file
```

It's not always the case that we write less number of line in ts and get more js code instead reverse is true , we write more in ts but it a safe code.



These are some of the types that are available in the typescript. Using ANY s type means we're not type restricting the typescript making the code more javascriptish.

In typescript typescript can automatically detect the types in some of the cases , this is known as type inferring in typescript

```
let userId : number = 12 //declaring a number

/*
we can do this but this is not the good practice
because it is too obvious and we're overusing our typescript
typescript is smart enough to detect that userId is of type number
even if we don't explicitly declare userId of type number
This is called type inference
*/

let userId2 = 12 //even in this case it detected that this variable is of
//number type and if i place a . like userId2. then we got only number
//methods in suggestion

let hero:string

function getHero(){
   return "thor"
}
hero = getHero()
//now here in above case it is necessary to infer the type as string

/*
In TypeScript, the noImplicitAny option is a compiler setting that helps catch
potential bugs by ensuring that the type of every variable and parameter is explicitly
declared or can be inferred by the compiler.
When this option is enabled, TypeScript will raise an error if it encounters a
variable or parameter whose type is implicitly any.

To enable noImplicitAny, you need to configure your tsconfig.json file. Here's how you
can do it:

Locate or create tsconfig.json: This file is typically found in the root of your
TypeScript project. If you don't have one, you can create it using the TypeScript
compiler (tsc) with the command tsc --init.
```

```
Modify tsconfig.json: Add or update the noImplicitAny setting in the compilerOptions
section of your tsconfig.json file. The relevant part of the configuration might look
like this:
*/


//Typescript's type inference is so smart that it detect type automatically from the
//context

let heros = ["h1","h2","h3"]

//here it is automatically detecting the type
//of hero as string and even the type of lis is guessed correctly by the lis
const lis = heros.map((hero)=>{
    return hero
})


//void means function will return void and will not return any
//kind of value
function consoleError() {
   console.log("tiwari");
}
//never means that the end point of the function will never be reached
function handleError() {
   console.log("never function ");
   throw new Error("Hey! I am a Error");
}
```

Typescript Objects

```
function course2():{}{
    return {}
}
//same same but different
function course():Object{
   return {}
}
let a = course()
console.log(a)


//the bad behviour of typescript


type User = {   //this is type alias
   name : String,
   pass : String
}

function createUser(user : User){
   console.log(user.name)
}

let newUser = {name:"Aakash",pass:"password" , email:"atiwari@ivp.in"}
createUser(newUser)  //ideally it should give error becuase we're passing an email but
this is not the case
//hence this is the bad behaviour of the typescript

createUser({name:"Aakash",pass:"password" , email:"atiwari@ivp.in"}) //here it will
give you error
```

Readonly and optional in Typescript :-

```
type User = {
   readonly _id: string,
   name : string,
   email : string
   salary?:number //optional for berojgaars
}

let myUser : User = { //initializing an object
   _id : "1",
   name : "Aakash",
   email : "atiwari@ivp.in"
}

myUser.name = "Aakash2"
//myUser._id = "2" //giving error because _id is readonly

//now suppose we need to make a type from 2 existing type then we can simply
//use ampercent sign for doing so.

type idProof = {
   aadharCard : string,
   passportNumber? : string
}

type completeUser = User & idProof

let finalUser : completeUser = {   //two properties are missing as they are optional
   _id : "1",
   name : "lassi",
   email : "email@google.in",
   aadharCard : "32493994990"
}
```

Tuples : -

```
/*
   A tuple is just a typed array with a predefined length and types
   for each index.
*/

let ourTuple : [number,boolean,string] //predfined with 3 length and types

ourTuple  = [5,false,'coding is god']
//ourTuple = [5,4,2] //error

//bad behaviour of tuple

ourTuple.push("12") //no error is there
```

Interfaces :-

```
interface User {
   name : string,
   class : number ,
   income? : number,
   //here we can give functions as well

   wantAdmission():boolean ,
   enroll(collegeId : string , sex : string) : boolean
}


const firstKid : User = {
   name : "Aakash",
   class : 12,
   income : 0,
   wantAdmission() {
       return false
   },
```

```
   enroll(id: "at",sex:"M"){  //type is auto infered so We do not have to redfine it
      console.log(id+sex)
       return true
   }
}


//program


// Define a function interface for a simple calculator
interface Calculator {
   // Function that takes two numbers and returns their sum
   add: (a: number, b: number) => number;

   // Function that takes two numbers and returns their difference
   subtract: (a: number, b: number) => number;

   // Function that takes two numbers and returns their product
   multiply: (a: number, b: number) => number;

   // Function that takes two numbers and returns their division
   divide: (a: number, b: number) => number;
 }
  // Implementing the Calculator interface
 const basicCalculator: Calculator = {
   add: (a, b) => a + b,
   subtract: (a, b) => a - b,
   multiply: (a, b) => a * b,
   divide: (a, b) => a / b,
 };
  console.log(typeof basicCalculator)
 // Example usage
 console.log(basicCalculator.add(5, 3));       // Output: 8
 console.log(basicCalculator.subtract(10, 4)); // Output: 6
 console.log(basicCalculator.multiply(2, 6));  // Output: 12
 console.log(basicCalculator.divide(20, 4));   // Output: 5

 export {}
```