16 December 2025 09:09

1. DOM - Tree structure created by Browser using HTML
2. Browser RENDERS the UI as per the DOM
3. Modify DOM in dynamic html - using the "document "API of JS
4. JS uses interpreters
5. Data types in JS are INFERRED
6. Default values in function parameters
   Add(n1=10,n2=10)
7. Rest parameters in functions
   pass variable number of args to the function
   Add(…nums) //nums is the REST param
   {
   Nums is processed as an array inside the function
   }

   Add(12,13,13,14)
8. Passing functions as parameters
9. FAT arrow functions - assigned to variables
10. foreach, filter, findIndex - Synchronous callback function
    CALLBACK function -- a function passed to another function
    Receiving function is responsible for calling the callback
    receiving_func(callback)
    {
    Callback()

    }

11. Functions returning functions ---
    a. Defined a function within a function
    b. Returned function returns with the CLOSURE (data of the outer context)

    factory(p1,p2)
    {
    Function incrementor()
    {
    //uses the p1 and p2
    }

    Return incrementor
    }

12. Spread operator = to copy objects like arrays and json
    Let a =[1,2,3]
    Let b = […a]

    Let obj = {x:12,y:13}
    Let copy = {…obj }

    //When there is CONTAINED object
    Let obj2 = { x:12,y:"ff", data: { q:13,p:14} }

    Let copy2 = { …obj2 } // SHALLOW COPY

Let copy2 = { …obj2, data: {…obj2.data} } //DEEP COPY

_____

13. Destructuring = convenience

let [x,y] = arr // x and y are variables that will get 0th 1th element

let {x,y} = obj // x and y are properties that are assigned to x and y variables

_____

Typescript = higher level type checking that works on top of JS

1. Let x : string
2. Let x : string | number }}} UNION
3. let x : string | undefined
4. func1(a: number , b:string) : number
5. Higher order types / User defined types

```
interface User
    {
        Name:string
        Age?:number
    }
```

6. let person : User = { Name:"rrr" }

7 function f1(u : User)

```
    {
        u.Name
        u.age
        u.city //CANT be accessed - its not part of User type

    }
```

TS =========> TSC ======>JS

JS ==========Interpreter ====>RUN

```
//DEFINE A TYPE
interface MyUser
{
   name:string
   age? : number  //age is optional
}
interface Resident extends MyUser
{
   adhaar:string
}
//function greet(user:any) {
function greet(user:MyUser){
  return "Hello " + user.name +"  "+user.age //+user.city
}
//type of person is INFERRED
const person  = { name: "Prachi", age: 25 , city:"pune"};
console.log(greet(person));
let p2 ={name:"priya"}
console.log(greet(p2))
//type of p3 is fixed to Resident
let p3 : Resident = {name:"prach"
                ,age:12,
               adhaar:"1234455",
```

```
                }
```

Aliasing in typescript ----

type keyword to PRODUCE different data types

```
type User = {
  id: number;
  name: string;
  email?: string; // optional property
};
let u : User={id:12,name:"qqq",email:"a@abc.com"}
type mydata = string|number|boolean
let value :mydata
value = "www"
value=12
value=true
//value=[]   //Not allowed
```

```
//SPECIFY the FUNCTION PROTOTYPE
type MathOperation = (a: number, b: number) => number;
interface Calculator
{
    num1:number
    num2:number
    calc : MathOperation
}
let mycalc :Calculator ={ num1: 10,
                    num2:20,
                    calc:(a,b)=>{ return a+b}
                  }
function greet( callback:(n:string)=>string) //PROTOTYPE of CB
{
   let s:string = callback("prachi")    //CALL to CB
   console.log(s)
}
greet((a)=>{ return "Good Morning "+a}) //IMPLEMENTATION of CB
```

Generics ----

Stack
- Array
- Push
- Pop
- showAll

```
interface Stack1
{
    Elements: number[]
    Push: (a:number)=> void
    Pop: () =>number
    showAll: ()=>void
}

interface Stack2
{
```

```
        Elements: string[]
        Push: (a:string)=> void
        Pop: () =>string
        showAll: ()=>void
    }
```

Generics can help us create a data type that is flexible to accommodate Stack of number, string, Invoice, Book ,……

Generics have a PLACEHOLDER = Formal Type Parameter

In this Example T is the placeholder

```
interface Stack<T>
{
        Elements: T[]
        Push: (a:T)=> void
        Pop: () =>T
        showAll: ()=>void
}

let mystack : Stack<string>    //string is the ACTUAL type parameter
let mystack:Stack<User>  //  User is the ACTUAL type parameter
```

```
//GENERICS
//PLACEHOLDER  = T
function identity<T>(value: T): T {
  return value;
}

 //ACTUAL TYPE PARAMETER  = string
let s =identity<string> ("pluto").substring(0,2).toUpperCase()
console.log(s)
```

```
function firstElement<T>(arr: T[]): T|undefined{
  return arr[0];
}


let n:number|undefined = firstElement<number>([12,13,19])
let s:string |undefined= firstElement<string>([])
 if( s == undefined)
    console.log(n)
else
  console.log(n,s.toUpperCase())
```

Literal = we can specify a particular value(s) should be assigned to a variable

```
//literal type
type weekend = "SUNDAY" | "SATURDAY"
let day :weekend
day = "SATURDAY"
day ="SUNDAY"
day ="MONDAY"
```

```
enum winter {
   oct,
   nov,
   dec,
   jan,
   feb
}

let m : winter = winter.dec  //dec is the numeric property of enum -
if(m == winter.dec)
{
   console.log("year end")
}
```

```
//DATA TYPE DECLARATION
enum winter {
   oct,
   nov,
   dec,
   jan,
   feb
}

let m : winter = winter.dec  //dec is the numeric property of enum -
if(m == winter.dec)
{
   console.log("year end")
}
console.log(m)

enum myop { plus,minus,mul}
function calculator(num1:number,num2:number,op:myop):void
{
   switch(op)
   {
      case myop.plus : console.log(num1+num2);break;
      case myop.minus : console.log(num1-num2);break;
      case myop.mul : console.log(num1*num2);break;
   }
}
//should I pass "+"  or  "PLUS" or "plus"
calculator(10,20,myop.plus)
```

Summarize a class in typsescript

1	use class keyword
2	class A extends B // where B is another class
3	class A implements I1, I2 // where I1 and I2 are interfaces

4	class A
	{
	static p1 : number //access using classname
	p2 : number //non static property , access using "this"
	}

5	private ,public and protected access specifiers

6	Declare a property in the constructor parameter list also
7	You can have only one constructor
8	You can use super keyword to access super class constructor or functions

---

Angular Framework

1. Get the angular CLI = ng
   npm install -g @angular/cli

2. Check
   ng version

3. Change directory to the folder where you want to download the template
   cd project

4. ng new earth(this is the project name)

5. cd earth
6. npm start (start the dev web server on port 4200)
7. open a browser type http://localhost:4200 and see the angular page of earth project

A node project -- npm init

1. **node_modules** folder = it has all the libraries/packages used by the app
   Whenever u say npm install --- that lib is added here
2. package.json , package-lock.json } for DEPENDENCIES and versions

A typescript project ---- ts init
Tsconfig

1. to check or specify ECMA script version - Javascript version
2. to control checking levels --- set different checks to true or false

---

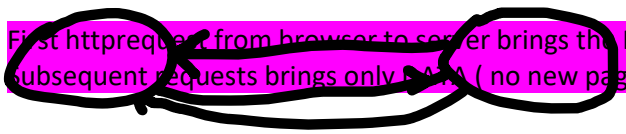
Multi page app = JSP/servlet , PHP , python template generators , ASP

browser web serverE

Every httprequest leads to a new page

Angular is mainly used for SPA = Single Page App
There is ONLY one HTML

First httprequest from browser to server brings the HTML +JS+CSS+images
Subsequent requests brings only JSON (no new pages come)



Web application =
MUST have a web server where the project is deployed
MUST have a web client - browser

The Angular template that we have downloaded
has a dev server embedded into it that is started using npm start
Convenience web server to be used only while developing the app !!!

Angular delivers only one page = index.html

Index.html ----------> <app-root> </app-root> (angular bootstrap component)
|
This will be Resolved using main.ts
|
Generated html will be added within the body of index.html

Angular - Component based architecture

Component1 +
Component2 +
Component3 +…..}Integrates to final HTML (index.html)

Each component is made up of

1. View template = html fragment
2. CSS
3. Component class in the TS file (MODEL , CONTROLLER)
4. Test bed = spec.ts

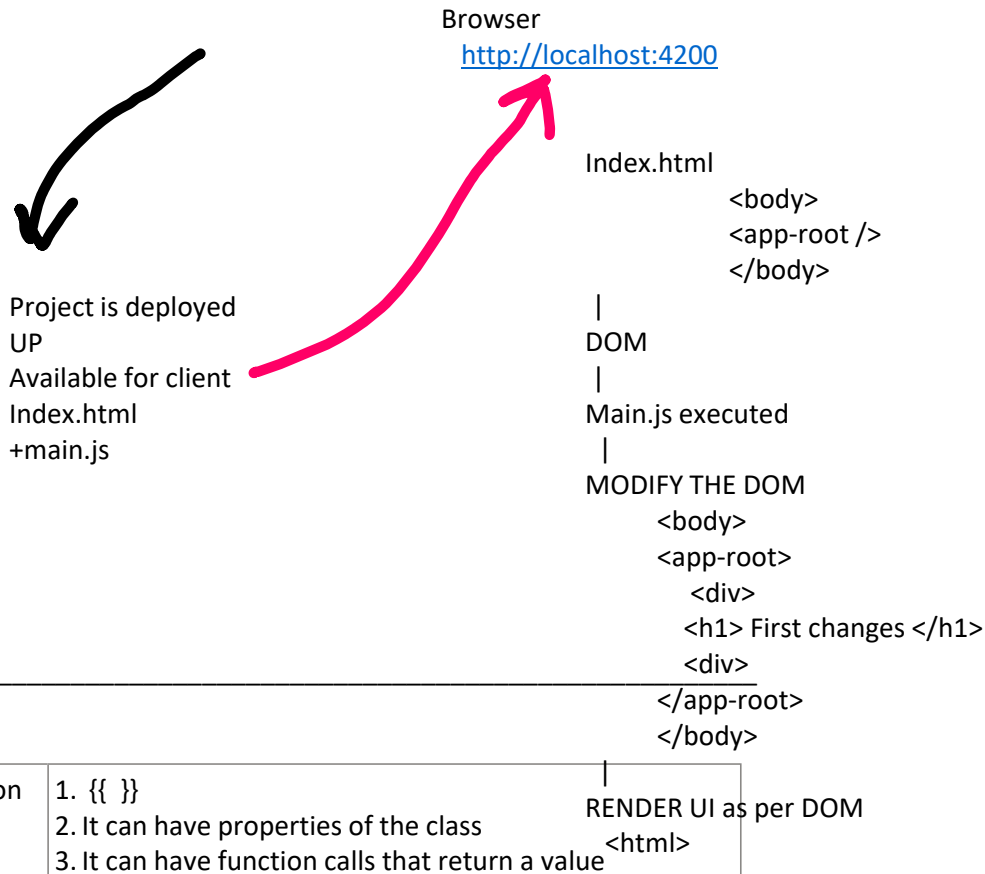
I change the app.html and app.ts
|
my angular BUILD gets generated -
generate a deployable project (index.html + main.js)
|
my project is deployed on web server (UP)
|
User accesses my page from the browser (http://localhost:4200)
|
the index.html + main.js goes to the browser/client
|
the DOM of the static index.html is created
|
main.js executes on the BROWSER
create the div tag , h1 tag and add it to the DOM
(then u see it in the inspect window of browser)
|
|
You see the app.html content on the browser UI RENDERED

Interpolation
= Way of sending data from MODEL to VIEW
Model = TS file
VIEW = html file

Selector = TAG = CUSTOM TAG / User defined TAG

VSCODE	
	Build folder created
.ts====>	Index.html + main.js

Web Server on 4200



Interpolation	1. {{ }} 2. It can have properties of the class 3. It can have function calls that return a value 4. It can have expressions that evaluate to a single value

Event Handling	1. user performs various actions on the UI 2. handle those actions - do something about the actions
	EVERYtime any event is handled the DOM is rerendered

Ex1 ----
Generate a new component counter
Add Increment button to the html
On clicking it show that a counter is incremented

To generate a new component
a. cd earth

b. ng g c counter
c. View = add increment button
d. Model = add the handler to increment

Index.html

```
<body>
  <app-root> </app-root> ==========> <app-counter> </app-counter>
</body>
```

EX2 - Add a decrement button that will decrement the counter using handler2

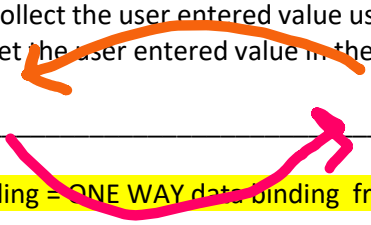
EX3 - write a commonhandler function in the counter component
And use the same handler for increment and decrement

write commonhandler2 - use the inbuilt object **$event**
And use the same handler for increment and decrement

EX4 -
MODEL ==========> VIEW - using interpolation
VIEW ----------EVENT ---------->MODEL handler
Using Event HANDLING - call the handler

MODEL VIEW

Add a textfield to get the value of counter from user
Collect the user entered value using $event in the MODEL
Set the user entered value in the counter property



Property binding = ONE WAY data binding from MODEL to VIEW
Property =
Interpolation shows the counter property {{ }}
Binding attribute of HTML element to Property of the component

```
<button disabled="true" >OK </button>   //Hardcoding

<button [disabled]="propName" >OK </button>
```

If the attribute is not put in the [] then RHS is a string
If the attribute is put in the [] then RHS is an expression to be

EX5 -
ng g c databinding

a. Add a textfield - disabled is true
Add a checkbox - if it is checked the textfield is enabled else the textfield is disabled

b. Write two CSS classes s1 and s2 in the css file
have a p tag
When the p tag is clicked toggle the class from s1 to s2

use (click) event
use property binding for class attribute of the p tag

c. A textfield number user can enter acquired marks
a dropdown list of total marks = 25, 50 , 75, 100
after user selects the total marks calculate the percentage
If the percentage is less than 35 show FAILED in RED color in a tag
Else show PASSED in GREEN color in the same tag
Change th color using STYLE BINDING
<p [style.color]="perc <35 ? 'red' : 'green' " >