TYPESCRIPT PRACTICE NOTES

Main.ts

// Data types

let myName:string = 'yaswanth'

// let year: number = 19;

let isLoading : boolean = true;

let album : any = 1999; // any type

let album1 : string | number | boolean // union type

let reg: RegExp = /\w+/g

// Arrays

let stringArr = ['yaswanth', 'is','good','boy']

let guitars = ['let start','yash',2003]

let mixedData = ['ETH',2003,true]

stringArr[0] = 'hi'

stringArr.push('hello')

guitars [0] = 1919

guitars [0] = 'yash'

guitars.unshift('hei')

let test=[]

let bands : string[] = []

bands.push('heeee')

//Tuple

let myTuple: [string, number, boolean] = ['yas',77,true]

let mixed = ['john',96,false]

mixed = myTuple

// myTuple = mixed -> It's give an error . The myTuple is fixed data types whereas in mixed or operator

//objects

let myObj:object ;

myObj = []

console.log(typeof myObj)

myObj = bands

myObj = {}

const exampleObj = {

    prop1:'yash',

    prop2:true,

}

exampleObj.prop1 ='sai'

type Guitarist = {

    name:string,

    active?:boolean,

    album: (string | number)[

    ]

}

let evh: Guitarist = {

    name : 'Eddie',

    active: false,

    album : [ 1984, 1989]

}

let jp: Guitarist = {

    name:'Jimmy',

    album:['I','II','III']

}

evh=jp

const greeGuitarist = (gutarist: Guitarist) => {

    return `Hello ${gutarist.name}`

}

console.log(greeGuitarist(jp))

// Type Aliases

type stringOrNumber = string | number

type stringOrNumberArray = (string | number)[]

type Guitarist1 = {

    name?:string,

    active:boolean,

    album:stringOrNumberArray

}

type userId= stringOrNumber

//Literal types

let myName1 : 'yash';

// myName1 = 'srinu' => error

let myName2 : 'yash' | 'syam' | 'subbu'

myName2 = 'syam'

myName2='subbu'

myName2 = 'yash'

// myName2 = 'sai' => error

//functions

const add = (a: number, b: number): number =>{

    return a+b;

}

const logMsg = (message : any): void => {

    console.log(message)

}

logMsg('hello');

logMsg(add(2,3));

let subtract = function(c: number,d: number): number {

    return c-d;

}

type mathFunction = (a: number,b: number) => number

// interface mathFunction {

//     (a: number, b: number): number

// }

let mutiply: mathFunction = function(c,d){

    return c \* d

}

logMsg(subtract(3,2))

logMsg(mutiply(4,9))

//optional Parameters

const addAll = (a: number,b: number,c?: number): number =>{

    if(typeof c !== 'undefined')

        return a+b+c

    return a+b

}

logMsg(addAll(2,3,4))

logMsg(addAll(2,3))

//default parameters

const sumAll = (a: number=20,b: number,c: number = 2): number =>{

    return a+b+c

}

logMsg(sumAll(2,3))

logMsg(sumAll(3,4,5))

// logMsg(sumAll(3)) => gives an error the default parameters set in the last of the function parameters

logMsg(sumAll(undefined,3))

//Rest parameters

const total = (a: number, ...nums: number[]): number =>{

    return a+nums.reduce((prev,curr)=> prev+ curr)

}

logMsg(total(2,3,4))

//never datatype

const createError = (errMsg: string): never =>{

    throw new Error(errMsg)

}

const infinite = ()=>{

    let i:number = 1

    while(true){

        i++;

        if(i>100) break

    }

}

//custom type guard

const isNumber = (value : any) : boolean =>{

    return typeof value === 'number' ? true : false

}

logMsg(isNumber(2))

// use of the never type

const numberOrString = (value : string | number ): string =>{

    if(typeof value === 'string') return 'string'

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

    return createError('This should never happen')

}

// chapter - 5 Assertions

type One = string

type Two = string | number

type Three = 'hello'

//convert to more or less specific

let a: One = 'hello'

let b = a as Two // less

let c = a as Three // more

let d=<One> 'world'

let e=<string | number> 'world'

const addOrConcat = (a: number, b: number, c:'add' | 'concat'): number | string =>{

    if(c === 'add') return a+b

    return ''+a+b

}

let myVal: string = addOrConcat(2,2,'concat') as string

// Be careful! TS see no problem - but a string is returned

let nextVal: number = addOrConcat(2,2,'concat') as number

(10 as unknown) as string

//The dom

const img = document.querySelector('img')!

img.src

const myImg = document.getElementById('#img') as HTMLImageElement

myImg.src

const nextImg = <HTMLImageElement>document.getElementById('#img')

copyright.ts

// const year = document.getElementById("year")

// const thisYear = new Date().getFullYear()

// year.setAttribute("datetime",thisYear)

// year.textContent = thisYear

// 1st Variation

// let year: HTMLElement | null

// year=document.getElementById("year")

// let thisYear: string

// thisYear = new Date().getFullYear().toString()

// if(year){

//     year.setAttribute("datetime",thisYear)

//     year.textContent = thisYear

// }

//2nd variation

const year =document.getElementById("year") as HTMLSpanElement

const thisYear: string = new Date().getFullYear().toString()

year.setAttribute("datetime",thisYear)

year.textContent = thisYear

