- PPT Slide

- Install typescript using terminal

npm i -g typescript

- Install Code Runner’

- Create Test.ts to demo below

- Test variable declaration var -> array -> tuples -> obj

- Just run with code runner, don’t need live server now

[Code to Demo - Pastebin.com](https://pastebin.com/i03rZiXJ)

[Workshop Walkthrough - Pastebin.com](https://pastebin.com/GhjDUdGF)

- Done demo -> delete Test.ts

- Add index.html and style.css

[index.html - Pastebin.com](https://pastebin.com/jZ8AsMfJ)

[style.css - Pastebin.com](https://pastebin.com/LpAeJS57)

- Install Live Server just for more convenient to compile

- Create app.ts -> console.log(“Hello World”) -> can’t compile

- Convert to .js using terminal

tsc app.ts

Then tsc app.ts -w to track the file

- Go to browser check console -> Done setup

- It will show error when app.ts and app.js open at the same time, just close app.js

- Continue with app.ts -> create class Book -> explain class and obj

Old way:

class Book{

    readonly title: string;

    private author: string;

    public year: number;

    constructor(title: string, author: string, year: number,

    ){

        this.title = title;

        this.author = author;

        this.year = year;

    }

    toString() {

      return `Title: ${this.title}, Author: ${this.author}, Publish Year: ${this.year}`;

    }

  }

New way:

class Book{

    // readonly title: string;

    // private author: string;

    // public year: number;

    constructor(

      readonly title: string,

      private author: string,

      public year: number,

    ){}

    toString() {

      return `Title: ${this.title}, Author: ${this.author}, Publish Year: ${this.year}`;

    }

  }

Class: A class in TypeScript is a blueprint for creating objects. It serves as a template for creating multiple instances (objects) that share the same structure and behavior.

Object: An object in TypeScript is an instance of a class. Each object has its own unique set of property values, but they all follow the structure defined by their class

Explain 3 access modifier types (public, private, readonly)

public: can access & change directly from outside

private: only access & change through methods in that class

readonly: can access from outside but cannot change value at all (inside + outside cuz it’s readonly), the property can only be assigned a value once, either in the constructor or at the point of declaration, and cannot be modified thereafter.

- JS doesn’t have OOP -> less secure

Doesn’t have readonly, private, public -> needs to use closure (create a function and define var in that function) or naming convention (have the underscore \_ before var name, just for dev to know, not work)

- This is shortcut, we put all parameters into it, it will automatically assign the value, unlike JS (open .js to compare)

- Function in TS is a type, function is normal function like JS

- Do example then check console

const book1 = new Book('Doraemon','Fujiko',1969);

console.log(book1);

console.log(book1.title);

// book1.title = 'Happy';

// console.log(book1.author);

book1.year = 2002;

console.log(book1.year);

- Variables declared with let are mutable, meaning their values can be reassigned.

- Create same thing for audiobook (copy paste)

const book1 = new Book('Doraemon','Fujiko',1969);

console.log(book1);

  class AudioBook{

    constructor(

      readonly title: string,

      private author: string,

      public year: number,

    ){}

    toString() {

      return `Title: ${this.title}, Author: ${this.author}, Publish Year: ${this.year}`;

    }

  }

  const audiobook1 = new AudioBook('Fallen Palace','Ming',2010);

  console.log(audiobook1);

- Add all DOM to link with .html

// form

const form = document.querySelector('.new-item-form') as HTMLFormElement;

// inputs

const type = document.querySelector('#type') as HTMLSelectElement;

const title = document.querySelector('#title') as HTMLInputElement;

const author = document.querySelector('#author') as HTMLInputElement;

const year = document.querySelector('#year') as HTMLInputElement;

If you don’t specify HTMLInputElement, TS only knows as element

- Add Event Listener (the same as JS so don’t need to explain much)

form.addEventListener('submit', (e: Event) => {

    e.preventDefault();

    console.log(type.value,title.value,author.value,year.valueAsNumber);

  });

Use valueAsNumber to make sure it’s a number

- Split file for better work flow

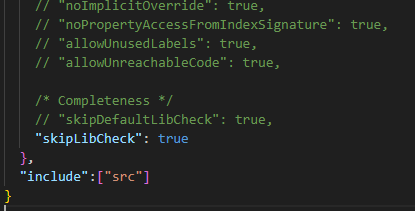
- Create public folder, move everything except .ts

- Create src folder, move .ts to it

- Open terminal, type tsc –init

- Open tsconfig.json

- Change "rootDir": "./src" and "outDir": "./public"



Add this so that it only tracks file in src. Files that are outside src won’t be convert to .js

- Go to tsconfig.json change "module": "ES2022", "target": "ES6"

Now go back to terminal type tsc -w

It automatically convert all .ts file in src to .js and put in public

- Close current browser, open live server again cuz html file change path

- Create classes folder in src, create Book.ts and AudioBook.ts, copy paste from app.ts

export class Book{

    // readonly client: string;

    // private details: string;

    // public amount: number;

    constructor(

      readonly title: string,

      private author: string,

      public year: number,

    ){}

    toString() {

      return `Title: ${this.title}, Author: ${this.author}, Publish Year: ${this.year}`;

    }

  }

export class AudioBook{

    constructor(

      readonly title: string,

      private author: string,

      public year: number,

    ){}

    toString() {

      return `Title: ${this.title}, Author: ${this.author}, Publish Year: ${this.year}`;

    }

  }

- Import them back to app.ts

- Go to app.js and to check the import (add .js to the import) -> Check console

- Create interface folder in src, create HasToString.ts

export interface HasToString {

    toString(): string;

  }

- Go to Book.ts and AudioBook.ts -> implements HasToString

- Go to app.ts import HasToString

- Create ListItem.ts in classes folder

import { HasToString } from "../interfaces/HasToString";

export class ListItem {

  constructor(private container: HTMLUListElement){}

  render(item: HasToString, heading: string, pos: 'start' | 'end'){

    const li = document.createElement('li');

    const h4 = document.createElement('h4');

    h4.innerText = heading;

    li.append(h4);

    const p = document.createElement('p');

    p.innerText = item.toString();

    li.append(p);

    if(pos === 'start'){

      this.container.prepend(li);

    } else {

      this.container.append(li);

    }

  }

}

To render to html, same as JS

- Go to app.ts import ListItem and add below

// list template instance

const ul = document.querySelector('ul')!;

const list = new ListItem(ul);

Since we have interface, now we’re going to use it to separate Book and Audio Book, then convert to html as well

form.addEventListener('submit', (e: Event) => {

    e.preventDefault();

  let data: HasToString;

  if (type.value === 'book') {

    data = new Book(title.value,author.value,year.valueAsNumber);

  } else {

    data = new AudioBook(title.value,author.value,year.valueAsNumber);

  }

  console.log(data);

  list.render(data,type.value,'end');

});

If any errors happen, check app.js, always add .js at import