Building an Application with TypeScript

You Already Know

Course(s):

1. An Introduction to TypeScript

1. Hands-On TypeScript 3 and Angular 7 for Web Development

1. Design Patterns in TypeScript

Recap

- List the features of ES6
  - Arrow functions
  - Working with *const, var,* and objects

- List the types of primitive and non-primitive data types
  - Boolean, number, and string
  - Arrays, tuples, and objects

- Explain the importance of Namespaces, Modules, and Generics
  - Namespace imports
  - Module resolution
  - Generic types and arrays

Recap

- Get started with TypeScript and Angular
  - Setting up Node.js and NPM
  - Setting up the TypeScript environment

- Dive into TypeScript 3
  - Data types, interfaces, and classes
  - Namespaces, modules, and functions

- Dive into Angular 7
  - Create a project
  - Create custom components
  - Add Bootstrap to Angular project

Recap

- Explain the implementation of JavaScript libraries with TypeScript
  - Import a library
  - Manage TypeScript definition files

- Explain the workflow of TypeScript
  - Working with *tsc* and the *tsconfig* file
  - Working with Gulp and Webpack

- Explain design patterns used in TypeScript
  - Creational design patterns
  - Dependency Injection
  - Structural design patterns
  - Behaviorial Patterns

A Day in the Life of a MEAN Stack Developer

It’s another day in Joe’s life as a MEAN Stack Developer in Abq Inc. His company acknowledges and appreciates his commitment and progress. He has diligently completed all the projects assigned to him.

He has now been assigned to an important project.

He has been asked to develop a cart management application using TypeScript. This application should be able to add and delete items in the cart.

In this lesson, we will learn how to solve this real-world scenario to help Joe complete his task effectively and quickly.

Learning Objectives

By the end of this lesson, you will be able to:

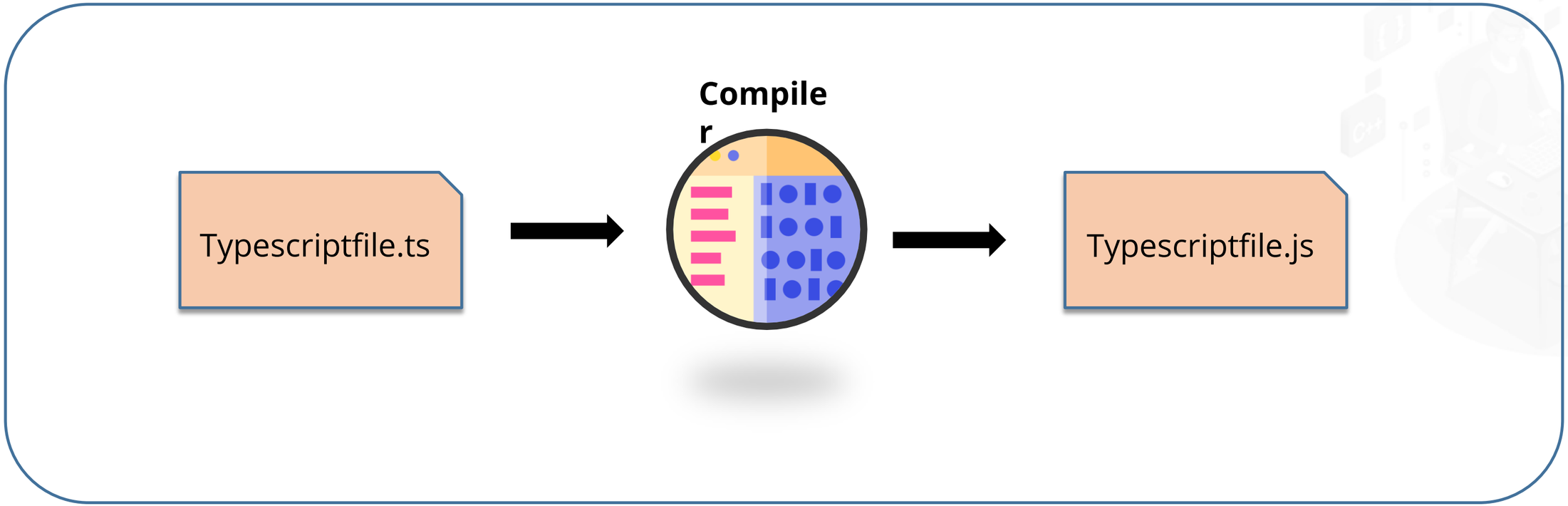
- List the features of TypeScript
- Set up a development environment
- Integrate TypeScript with build tools
- Migrate a JavaScript application to TypeScript
- Build an application that manages carts

TypeScript Configuration

TypeScript: Introduction

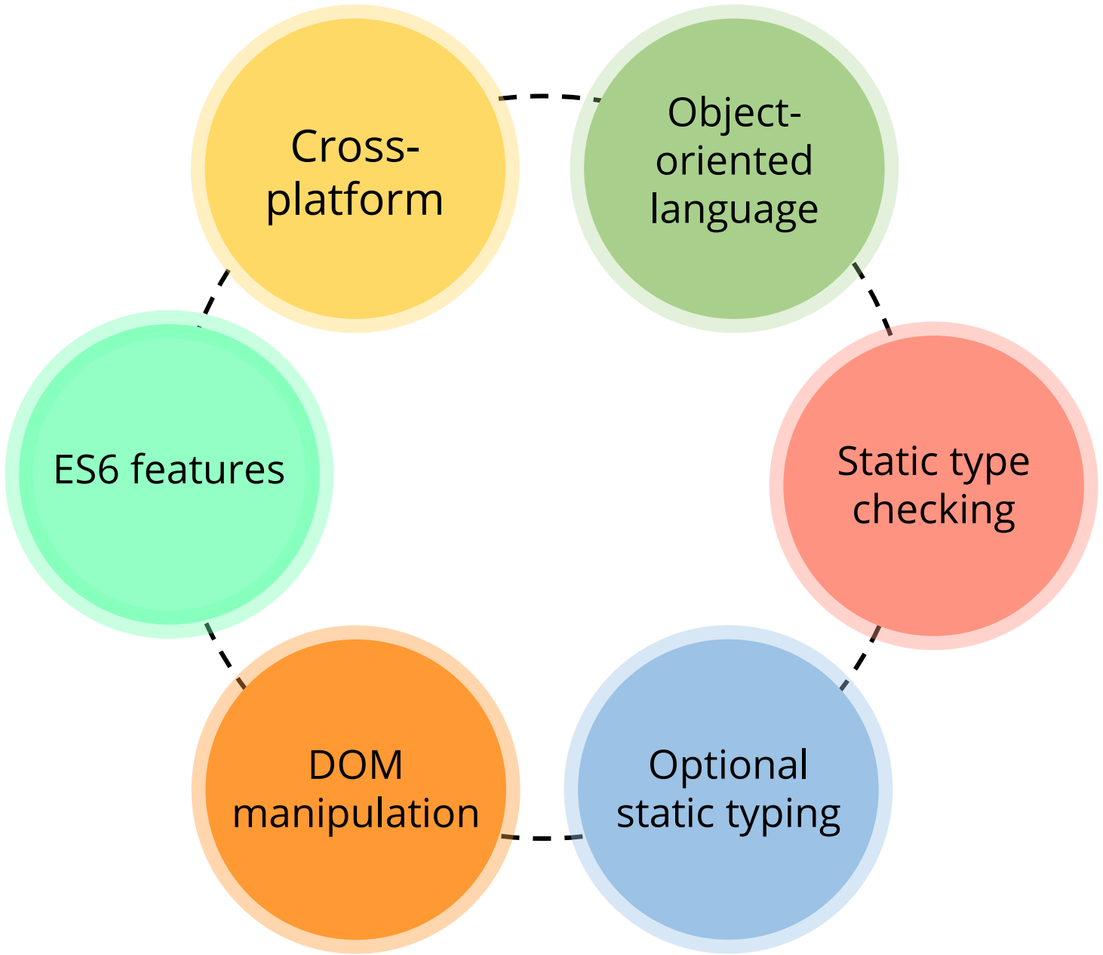
TypeScript is an open source, object-oriented programming language. It was developed by Microsoft under the Apache 2 license.

TypeScript cannot run directly on browsers. It requires a compiler to compile into JavaScript. A TypeScript file has a *.ts* extension.



TypeScript: Features and Advantages

Popular features of TypeScript:



Advantages of TypeScript:

- It is an open source, object-oriented language with continuous development
- It runs on any browser with JavaScript engine
- It supports integration with various tools like Grunt to automate the workflow
- Its code can be called from a JavaScript code
- It supports JavaScript features of ECMAScript 2015 along with ES7

TypeScript: Development Environment

Two popular ways to install TypeScript are:

- **Using Node Package Manager (NPM)**

Node.js and an IDE are required to use TypeScript in a project. TypeScript can be installed via npm by executing the following command:

```
> npm install –g typescript
```

Check the TypeScript version executing the command: ***tsc -v***

- **Using the TypeScript plugin**

TypeScript plugin is available for various popular IDEs. A few of them are:

- Atom-TypeScript Plugin
- TypeScript IDE for Eclipse
- TypeScript Plugin for Eclipse

TypeScript Plugin with Atom

Atom is an open source code editor for Linux, Windows, and macOS with supported plugins.

Steps to use the TypeScript plugin:

- Install the Atom editor
- Install the required dependencies
  - Option 1: Install *atom-ide-ui* package
  - Option 2: Install the following packages:
    - *linter*
    - *linter-ui-default*
    - *hyperclick*
    - *intentions*
- Start Atom and open a TypeScript file

TypeScript Plugin with Visual Studio Code

Visual Studio Code is an editor developed by Microsoft for Windows, Linux, and macOS. It supports TypeScript files. A TypeScript compiler is required to transpile code to JavaScript.

- Install the TypeScript package via npm:

```
npm install –g typescript
```

- Install TypeScript package locally via npm:

```
npm install –save-dev typescript
```

tsconfig.json File

The *tsconfig.json* file indicates that the current directory is the root of a TypeScript project. A TypeScript project can be compiled in any one of the following methods:

- By invoking tsc without input files
- By invoking tsc without input files and a *--project* command-line option that specifies the directory path which contains a tsconfig.json file

```
{
        "extends": "./configs/base",
        "files": [
                "main.ts",
                "supplemental.ts"
                ]
}
```

Compiler Options

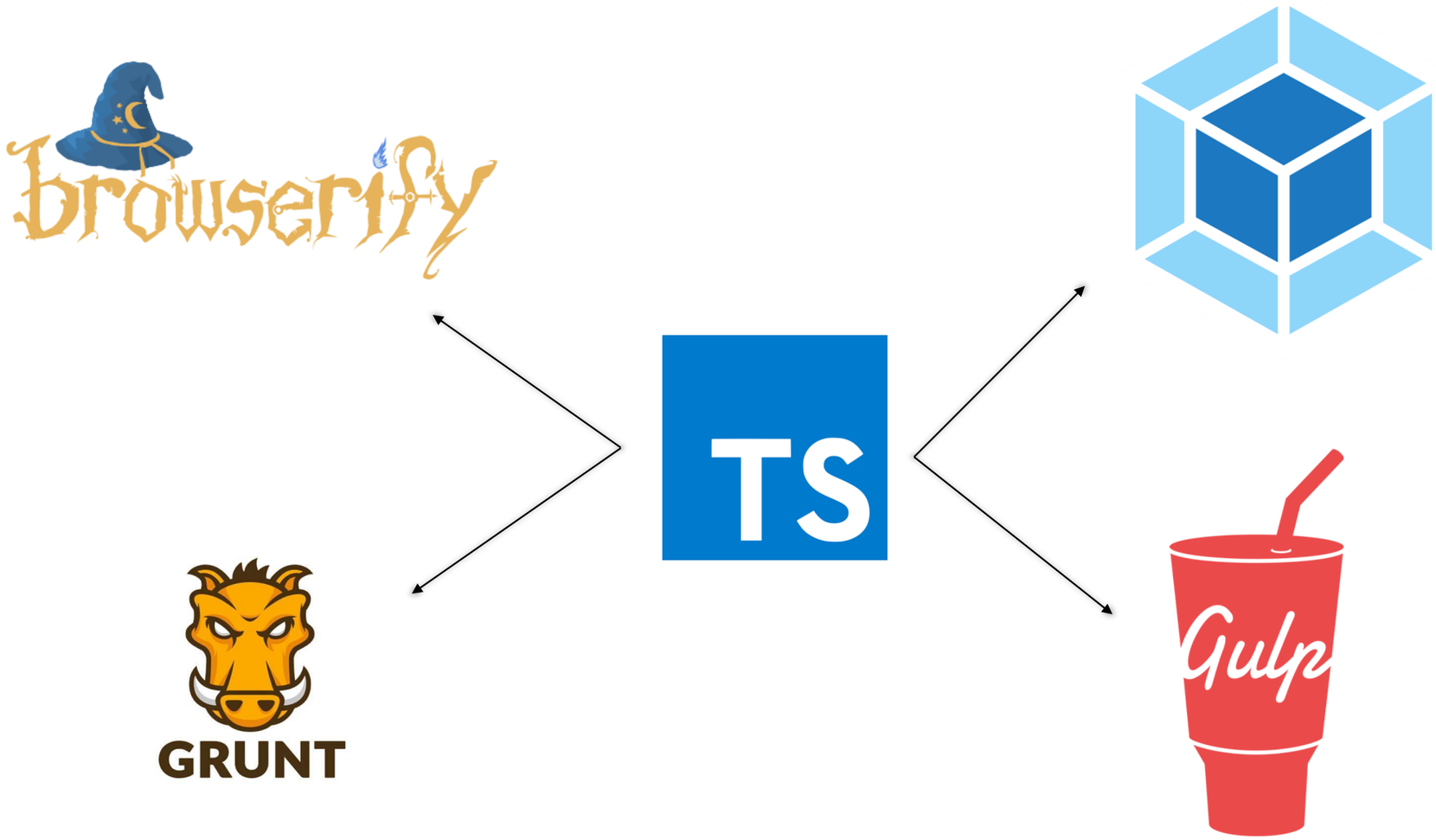
Option	Type	Default	Description
--allowJs	boolean	FALSE	Allows JavaScript files to be compiled
--allowUnreachableCode	boolean	FALSE	Does not report errors on unreachable code
--allowUnusedLabels	boolean	FALSE	Does not report errors on unused labels
--alwaysStrict	boolean	FALSE	Parse in strict mode and adds **use strict** for each source file
--charset	string	"utf8"	Is the character set of input files
--checkJs	boolean	FALSE	Reports errors in .js files and is used in conjunction with --allowJs

Compiler Options

Option	Type	Default	Description
--composite	boolean	TRUE	Ensures TypeScript can determine where to find output of the referenced project to compile it
--declaration	boolean	FALSE	Generates corresponding .d.ts file
--declarationMap	boolean	FALSE	Generates a source map for each corresponding .d.ts file
--diagnostics	boolean	FALSE	Shows diagnostic information
--disableSizeLimit	boolean	FALSE	Disables size limitation on JavaScript project
--downlevelIteration	boolean	FALSE	Provides full support for iterables in for..of and spread and destructures when compiler targets ES5 or ES3

TypeScript with Build Tools

Some of the popular build tools that can be integrated with TypeScript are:



TypeScript with Browserify

Browserify is an open source JavaScript tool used to write Node.js-type modules that compile for use in browsers.

Install *tsify* via npm using the following command:

```
> npm install tsify
```

Use the following command to compile the source code and store the result in another file using the following command:

```
> Browserify main.ts -p [ tsify --noImplicitAny ] > filename.js
```

TypeScript with Gulp

Gulp is an open source JavaScript kit used as a build system. It is built on Node.js and npm. It is used for automation of repetitive tasks such as minification, linting, and optimization.

Install *gulp-typescript* via npm using the following command:

```
> npm install gulp-typescript
```

Add the *ts* task in *gulpfile.js.*

Run *gulp* at the command line to compile TypeScript code.

```
var gulp = require('gulp');
var ts = require('gulp-typescript');

gulp.task('default', function () {
    return gulp.src('src/**/*.ts')
        .pipe(ts({
            noImplicitAny: true,
            outFile: 'output.js'
        }))
        .pipe(gulp.dest('built/local'));
});
```

TypeScript with Grunt

Grunt is a JavaScript-based task runner tool used to perform tasks such as minification, unit testing, linting, and compilation. Grunt-ts is an npm package that manages TypeScript compilation in GruntJS build scripts.

Install *grunt-ts* via npm using the following command:

```
> npm install grunt-ts
```

Add the *ts* task in *Gruntfile.js.*

Run *grunt* at the command line to compile TypeScript code.

```
module.exports = function(grunt) {
  grunt.initConfig({
    ts: {
      default : {
        tsconfig: './tsconfig.json'
      }
    }
  });
  grunt.loadNpmTasks("grunt-ts");
  grunt.registerTask("default", ["ts"]);
};
```

TypeScript: Dos and Don'ts

Dos	Don’ts
Use *void* to return an ignored value for callbacks	Use of *any* for an ignored value for callbacks
Use of callback parameters which are not optional	Use of optional parameters in callbacks
Use of a single overload function	Use of separate overload functions
Use of union types for overload functions	Use of overload functions that differ by type in only one argument position

TypeScript Configuration with Grunt

Duration: 30 min.

Problem Statement:
You are given a project to configure TypeScript with Grunt.

ASSISTED PRACTICE

Assisted Practice: Guidelines to Configure TypeScript with Grunt

1. Set up Grunt environment with TypeScript.
2. Generate a .js file with server.ts and Gruntfile.js.

JavaScript to TypeScript

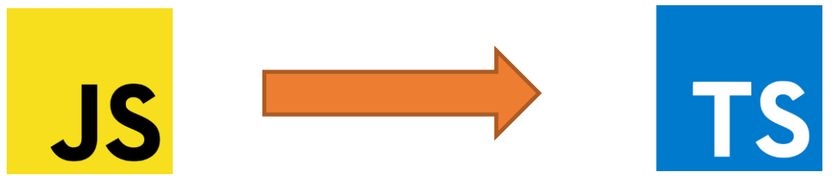
Differences: JavaScript and TypeScript

JavaScript	TypeScript
Data binding is not available in JavaScript	Data binding is possible using types and interfaces
Compiling the source code is not required	Source code should be compiled
It is easy to learn and implement	It is difficult to learn and implement
It is a scripting language	It is an object-oriented language
Static typing is not possible	Static typing is possible to implement

Best Practices of TypeScript

- Strict Configuration
- Callback Types
- Type Inference
- Use of Primitive Types
- Function Parameters

Migration: JavaScript to TypeScript



Steps to migrate from JavaScript to TypeScript:

- Add a tsconfig.json file
- Integrate with a build tool
- Transfer all .js files to .ts files
- Debug for errors
- Use third-party JavaScript libraries

TypeScript: Basic Types

TypeScript supports these popular data types:

- **Boolean:** The data type that has two values, true and false
- **Number:** The data type supporting floating point values
- **String:** The data type to interpret a string value within double or single quotes
- **Array:** The data type used to store multiple values in a single variable
- **Tuple:** The data type used to store different types of values in an array
- **Any:** The data type used to opt-in and opt-out of type checking during compilation
- **Void:** The data type used to indicate absence of a type
- **Never:** The data type used to indicate types of values that never occur

TypeScript: Advanced Types

TypeScript supports these advanced types:

- **Intersection:** The data type used to combine multiple types into one
- **Union:** The data type used to describe a value that can be one of several types
- **String Literal:** The data type that allows users to specify the precise value a string should have
- **Conditional:** The data type used to add the ability to express non-uniform type mapping

Migration: JavaScript to TypeScript

Duration: 25 min.

Problem Statement:
You are given a project to demonstrate migration of the source code or a project from JavaScript to TypeScript.

ASSISTED PRACTICE

Assisted Practice: Guidelines to Migrate from JavaScript to TypeScript

1. Initialize the tsconfig.json file.
2. Set up the TypeScript environment.
3. Migrate from JavaScript to TypeScript.

TypeScript Decorators

TypeScript: Decorators

Decorators are used to add annotations and meta-programming syntax for class members and declarations. A decorator is available as an experimental feature that may change in future releases.

Decorators can be implemented in one of the following ways:

- Enabling the *experimentalDecorators*:

```
> tsc --target ES5 -- experimentalDecorators
```

- Modifying the *tsconfig.json* file:

```
{
        “compilerOptions” : {
        “target” : ”ES5”,
        “experimentalDecorators” : true
        }
}
```

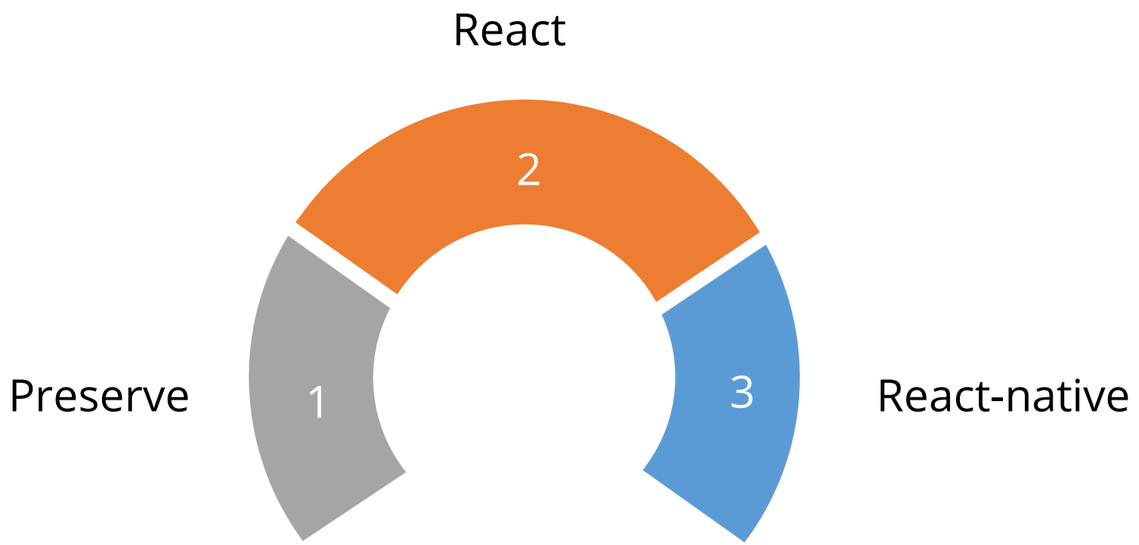
TypeScript: JSX

JSX is an embeddable XML-like syntax. TypeScript supports type checking and compiles JSX directly to JavaScript.

JSX can be implemented using the following steps:

- Save a file with *.tsx* extension
- Enable the *jsx* option

There are three JSX shipping modes, namely:



TypeScript Decorators

Duration: 20 min.

Problem Statement:
Write a program in TypeScript to implement decorators.

ASSISTED PRACTICE

Assisted Practice: Guidelines to Implement Decorators

1. Write a function in TypeScript to implement decorators.

2. Compile the code.

3. Push the code to GitHub repositories.

Inheritance, Interfaces, and Generics

TypeScript: Interfaces

- Interface is a structure that defines the contract in your application. It defines the syntax for classes to follow. Classes that are derived from an interface must follow the structure provided by their interface.
- The TypeScript compiler does not convert interface to JavaScript. It uses interface for type checking. This is also known as "duck typing" or "structural subtyping". An interface is defined with the keyword interface and it can include properties and method declarations using a function

Example:

Example: Interface

```
interface IEmployee {
empCode: number;
empName: string;
getSalary: (number) => number; // arrow function
getManagerName(number): string;
}
```

TypeScript: Classes

ECMAScript 2015, also known as ES6, allows programmers to use classes in TypeScrip

Example:

```
class message_class {

        msg: string;
        constructor (message: string) {
        this.msg = message;
        }

        body() {
        return "hello,"+this.msg;
        }

}

let messageBody = new body(" User");
```

TypeScript: Generics

TypeScript supports generics where a component can be created that can work over different types.

Example:

```
class message_class <T,U>{

        private key: T;
        Private key: U;

        setKeyValue(key: T, val: U): void {
        this.key = key;
        this.val = val;
        }

        display():void {
        console.log(`key =$ {this.key}, val = ${this.val}` );
        }

let values = new message_class <number, string>();
values.setKeyValue(1,"Tomatos");
values.display();
```

Inheritance and Constructors

Duration: 20 min.

Problem Statement:
Write a program in TypeScript to implement inheritance.

ASSISTED PRACTICE

Assisted Practice: Guidelines to Implement Inheritance

1. Write a function in TypeScript to implement inheritance.

2. Compile the code.

3. Push the code to your GitHub repository.

Getters and Setters

Duration: 20 min.

Problem Statement:
Write a program in TypeScript to implement getters and setters.

ASSISTED PRACTICE

Assisted Practice: Guidelines to Implement Getters and Setters

1. Write the function in TypeScript to implement getter and setter.

2. Compile the code.

3. Push the code to your GitHub repository.

Import and Export Modules

Duration: 20 min.

Problem Statement:
Write a program in TypeScript to import and export modules.

ASSISTED PRACTICE

Assisted Practice: Guidelines to Import and Export Modules

1. Write the function in TypeScript to implement import and export modules.

2. Compile the code.

3. Push the code to your GitHub repository.

Interfaces

Duration: 20 min.

Problem Statement:
Write a program in TypeScript to implement interfaces.

ASSISTED PRACTICE

Assisted Practice: Guidelines to Implement Interfaces

1. Write a function in TypeScript to implement interfaces.

2. Compile the code.

3. Push the code to your GitHub repository.

Generics

Duration: 20 min.

Problem Statement:
Write a program in TypeScript to implement generics.

ASSISTED PRACTICE

Assisted Practice: Guidelines to Implement Generics

1. Write a function in TypeScript to implement generics.

2. Compile the code.

3. Push the code to your GitHub repository.

Key Takeaways

- TypeScript is an object-oriented, portable programming language. It supports JavaScript libraries
- Visual Studio Code is an editor developed and Microsoft for Windows, Linux, and macOS which supports TypeScript files
- Decorator is used to add annotations and a meta-programming syntax for class members and declarations
- Typescript interfaces are responsible for naming the type checkers which focuses on the *shape* that values have

Cart Management Application

Duration: 30 min.

Problem Statement: Create an app where you can add and remove items in cart using TypeScript.

LESSON-END PROJECT

Before the Next Class

Courses: Angular

You should be able to:

- Build Angular components
- Understand Bootstrap
- Explain binding and events
- Explain dependency injection and services
- Work with directives, pipes, and forms
- Test an Angular app