|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Course Code** | **WEB FRAMEWORKS** | **L** | **T** | **P** | **C** |
| **IT 2404** | **3** | **0** | **2** | **4** |

#### Course Objectives:

* 1. To build scalable web applications using Angular
  2. To import and export functionalities of modules using Angular
  3. To create reusable UI components using React
  4. To manage state of the application more efficiently using React Hook
  5. To containerize the applications using Docker ad Kubernetes

#### UNIT I – ANGULAR V 12 9

Introduction to Angular – Typescript (Arrays, Functions, classes) – JS vs TS – Angular CLI Installation – Components – Data Binding – Routing on Angular - Directives

#### UNIT II – ANGULAR MODULES AND MATERIAL 9

Angular Modules – HTTP client, Forms Module – Angular Service Files – Dependancy Injection – Angular Material – Connecting Angular with Back End

#### UNIT III – REACT V 18 9

Introduction to React – Setting development environment – create app – JSX syntax – properties and states – components – React routing – API request

#### UNIT IV – REACT HOOKS 9

React Hooks – useState – useEffect – useCallback – useMemo – useContext – useReducer – Introduction to React Native

#### UNIT V – CONTAINERIZATION 9

Introduction to Image and Container – Docker – Containers – Docker Images, Docker file, Docker Network – Docker Compose - Kubernetes

#### 45 PERIODS

**PRACTICAL EXERCISES: 30 PERIODS**

1. Project – Create an angular app with n components and add routing
2. Project – Add functionalities, validation and database with above components
3. Project – Create Login System using React
4. Project – Create Flight Management system

#### COURSE OUTCOMES:

Upon completion of the course, students will be able to:

1. **CO1:** Build scalable web applications using Angular
2. **CO2:** Import and export functionalities of modules using Angular
3. **CO3:** Create reusable UI components using React
4. **CO4:** Manage state of the application more efficiently using React Hook
5. **CO5:** Containerize the applications using Docker ad Kubernetes

#### TEXT BOOKS:

* 1. Nate Murray, Felipe Coury, Ari Lerner, Carlos Taborda, “ The Ng book — The Complete Book on Angular”
  2. The Road to React, Robin Wieruch,2023.
  3. The Docker Book: Containerization is the new virtualization, James Turnbull, 2014.
  4. The Kubernetes Book, Nigel Poulton, 2023.

#### ONLINE RESOURCES:

1. https://angular.io/docs
2. https://react.dev/
3. https://react.dev/reference/react
4. https://docs.docker.com/
5. https://kubernetes.io/docs/home/

Introduction to Angular & Angular CLI Installation

Angular is an application-design framework and development platform for creating efficient and sophisticated single-page apps. It is built on TypeScript.

As a platform, Angular includes:

* A component-based framework for building scalable web applications
* A collection of well-integrated libraries that cover a wide variety of features, including routing, forms management, client-server communication, and more
* A suite of developer tools to help us develop, build, test, and update our code

## **Local development environment**

## **Step 1 - Identify the version of node.js that Angular requires**

Angular requires an active LTS or maintenance LTS version of Node.

From a Terminal window:

1. Run the following command: node --version
2. Confirm that the version number displayed meets the requirements.

## **Step 2 - Install the correct version of node.js for Angular**

## **Step 3 - Install the latest version of Angular**

With node.js and npm installed, the next step is to install the [Angular CLI](https://angular.io/cli) which provides tooling for effective Angular development.

From a **Terminal** window run the following command: npm install -g @angular/cli.

## **Step 4 - Install integrated development environment (IDE)**

1. [Visual Studio Code](https://code.visualstudio.com/)

## **Create a new workspace and an initial application**

Develop applications in the context of an Angular [workspace](https://angular.io/guide/glossary#workspace). A workspace contains the files for one or more [projects](https://angular.io/guide/glossary#project). A project is the set of files that make up an application or a library.

To create a new workspace and an initial project:

1. Ensure that you aren't already in an Angular workspace directory. For example, if you're in the Getting Started workspace from an earlier exercise, navigate to its parent.
2. Run ng new followed by the application name as shown here:

ng new angular-tour-of-heroes

1. ng new prompts you for information about features to include in the initial project. Accept the defaults by pressing the Enter or Return key.

ng new installs the necessary npm packages and other dependencies that Angular requires. This can take a few minutes.

ng new also creates the following workspace and starter project files:

* A new workspace, with a root directory named angular-tour-of-heroes
* An initial skeleton application project in the src/app subdirectory
* Related configuration files

The initial application project contains a simple application that's ready to run.

## **Serve the application**

Go to the workspace directory and launch the application.

cd angular-tour-of-heroes

ng serve --open

The ng serve command:

* Builds the application
* Starts the development server
* Watches the source files
* Rebuilds the application as you make changes

The --open flag opens a browser to http://localhost:4200.

You should see the application running in your browser.

## **Angular components**

The page you see is the application shell. The shell is controlled by an Angular **component** named AppComponent.

Components are the fundamental building blocks of Angular applications. They display data on the screen, listen for user input, and take action based on that input.

## **Make changes to the application**

Open the project in your favorite editor or IDE. Navigate to the src/app directory to edit the starter application. In the IDE, locate these files, which make up the AppComponent that you just created:

| FILES | DETAILS |
| --- | --- |
| app.component.ts | The component class code, written in TypeScript. |
| app.component.html | The component template, written in HTML. |
| app.component.css | The component's private CSS styles. |

When you ran ng new, Angular created test specifications for your new application.

Change the application title

Open the app.component.ts and change the title property value to 'Tour of Heroes'.

app.component.ts (class title property)

title = 'Tour of Heroes';

Open app.component.html and delete the default template that ng new created. Replace it with the following line of HTML.

app.component.html (template)

<h1>{{title}}</h1>

The double curly braces are Angular's *interpolation binding* syntax. This interpolation binding presents the component's title property value inside the HTML header tag.

The browser refreshes and displays the new application title.

Add application styles

Most apps strive for a consistent look across the application. ng new created an empty styles.css for this purpose. Put your application-wide styles there.

Open src/styles.css and add the code below to the file.

src/styles.css (excerpt)

/\* Application-wide Styles \*/

h1 {

color: #369;

font-family: Arial, Helvetica, sans-serif;

font-size: 250%;

}

h2, h3 {

color: #444;

font-family: Arial, Helvetica, sans-serif;

font-weight: lighter;

}

body {

margin: 2em;

}

body, input[type="text"], button {

color: #333;

font-family: Cambria, Georgia, serif;

}

button {

background-color: #eee;

border: none;

border-radius: 4px;

cursor: pointer;

color: black;

font-size: 1.2rem;

padding: 1rem;

margin-right: 1rem;

margin-bottom: 1rem;

margin-top: 1rem;

}

button:hover {

background-color: black;

color: white;

}

button:disabled {

background-color: #eee;

color: #aaa;

cursor: auto;

}

/\* everywhere else \*/

\* {

font-family: Arial, Helvetica, sans-serif;

}

# **Typescript vs Javascript**

Refer <https://radixweb.com/blog/typescript-vs-javascript>

With the increased popularity of TypeScript, we can consider it an enhanced version of JavaScript with extra features.

Talking about TypeScript, it is an open-source programming language for developing large and complicated applications. TypeScript was developed by one of the tech giants – Microsoft - in 2012 and the actual reason behind its creation was to handle large-scale applications.

Moreover, Angular is also using TypeScript for web development services. [**As per the study**](https://insights.stackoverflow.com/survey/2021#most-loved-dreaded-and-wanted-language-want), TypeScript and JavaScript are the second and third most popular language respectively that developers want to work with. Do you know why?

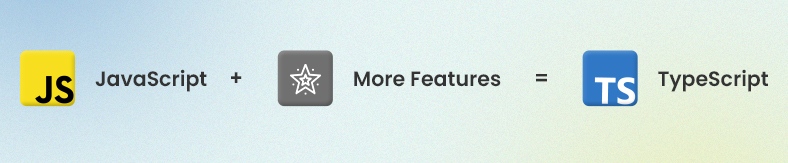
Actually, JavaScript is considered the primary scripting language for apps and web pages. Therefore, now, we can use JavaScript for both frontend frameworks like **[TezJS](https://tezjs.io/?utm_source=typescript-vs-javascript&utm_medium=radix-blog-text&utm_campaign=typescript-vs-javascript" \o "TezJS" \t "_blank)**, React, and backend with Node.Js and Deno frameworks.

But the actual question coming now is, was JavaScript developed for creating large and complex web apps? The answer is NO!

## **TypeScript vs JavaScript: The Real Differences**

If we consider both – TypeScript vs JavaScript, then every JavaScript code is valid in TypeScript. This means TypeScript is a superset of JavaScript.

In another way, we can say,



So, if you save your JavaScript programming file (.js) with a TypeScript (.ts) extension, it will execute perfectly fine. But that doesn’t mean that both languages - TypeScript and JavaScript are the same.

## **Why was TypeScript Developed Despite Having JavaScript?**

JavaScript language was introduced as a client-side programming language. But with the usage of [**JavaScript for web development**](https://radixweb.com/blog/best-javascript-frameworks-for-web-development), developers learned that it could also be considered a server-side programming language.

However, the JavaScript code became quite complex and heavy, too. Therefore, JavaScript could not be able to stand up to the expectations of an object-oriented programming language. As a result, JavaScript will never flourish as a server-side technology in the industry. This is why the TypeScript language was born to bridge this gap.

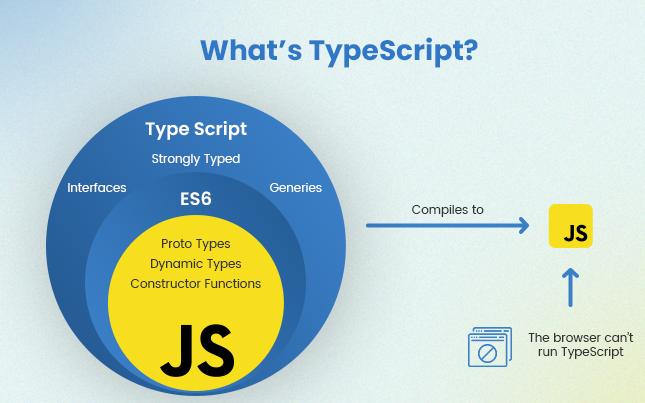
## **What’s TypeScript?**

If asked, “What’s better than JavaScript?”, you will surely get an answer – TypeScript.

Developed and maintained by Microsoft, TypeScript is an object-oriented, open-source programming language. It’s a superset of JavaScript, containing optional typing. Also, it compiles to plain JavaScript.

In a nutshell, TypeScript is a statically compiled programming language for writing clear and concise JavaScript code. It’s fulfilling the same purpose as JavaScript and can be used for both client-side and server-side applications. In addition, the libraries of JavaScript are also compatible with TypeScript.

TypeScript.



TypeScript is a programming language that supports both dynamic and static typing. It provides classes, visibility scopes, namespaces, inheritance, unions, interfaces, and many other features. Also, it offers comments, variables, statements, expressions, modules, and functions.

You can execute it on Node.Js or any other browser that supports ECMAScript 3 or its latest versions.

Since TypeScript is an enhanced version of JavaScript, all code of JavaScript is syntactically valid TypeScript. However, it doesn’t mean the TypeScript compiler can process all JavaScript:

*let a = 'a'; a = 1; // throws: error TS2322: Type '1' is not assignable to type 'string'.*

TypeScript provides files that can include type data from current object files, similar to how C++ header files define the creation of current object files. Hence, other apps can use the values defined in files just as TypeScript entities with statically typed values.

You will also find third-party header files for popular libraries like jQuery, D3.js, and MongoDB. There are also TypeScript headers for NodeJs elementary modules, which allow developers to create **[Node.Js development solutions](https://radixweb.com/nodejs-development" \o "Node.Js development solutions)** within TypeScript.

The compiler of TypeScript is assembled in JavaScript and written in TypeScript. It’s registered under Apache License 2.0.

After going through the TypeScript introduction, you must be wondering what would be the objective of adding static typing to JavaScript?

Well, we can clarify your doubts with the given statements:

* You can avoid hidden-ninja errors like the classic 'undefined' is not a function.
* Easy to refactor code without breaking it significantly.
* Orienting oneself in large-scale, complex systems is no longer a nightmare.

As per the study, TypeScript identifies around 15% of all JavaScript errors.

The freedom of dynamic typing frequently leads to errors, which not only reduces the developer’s efficiency but also can grind development due to the increasing expenses of adding new lines of code.

As a result, JavaScript is a poor choice for server-side code in organizations and large codebases due to its lack of types and compile-time error checks.

As their tagline suggests, TypeScript is JavaScript that scales.

## **Is TypeScript Suitable for Frontend or Backend?**

Since TypeScript is compiled to JavaScript, it is suitable for both the frontend and backend of app development.

Besides, JavaScript is a preferred programming language for the frontend of web pages and apps. As a result, TypeScript may be used for the same reason, but it also works well on the server-side for complicated and large-scale enterprise projects.

However, you can also use other [**top frontend frameworks**](https://radixweb.com/blog/top-front-end-frameworks-for-web-development) like React, Angular, Vue to build next-gen apps.

## **Types of TypeScript**

TypeScript contains various basic types, such as Number, Array, Tuple, Boolean, String, and many more. Well, some of these types are not available in JavaScript.

Furthermore, below are some other types which are expressivity of TypeScript:

### ****Any & Unknown****

A type called Any (anything that you wish) can cover unknown is its type-safe system. Here, any allows you to assign and JavaScript variable whenever you want to escape the type system. It’s widely used to describe incoming variables that haven’t been validated yet and whose type is unknown (for example, from third-party APIs).

Unknown is similar to Any, but it will not allow you to do anything with it unless it’s explicitly type-checked.?

### ****Void****

When there is no value returned, Void is used. Generally, it’s used for the return type of function that returns nothing.

### ****Never****

Never is the return type for something that should never happen, such as an exception-throwing function.

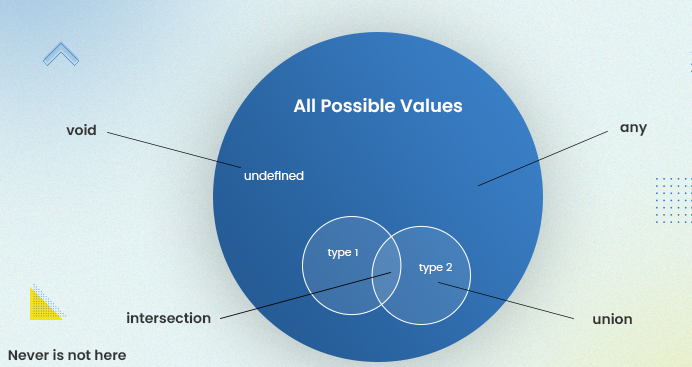
### ****Intersection & Union Types****

These types allow you to create custom types as per the logic.

Intersection types let you combine several basic types into one type. For example, if we create a custom type Person that contains *first\_name:string* and a *last\_name:string* . Well, you can say this as: I want my type to be this **and** that.

Union types allow you to type to take one of the various basic types. For example, if there’s a query that returns either *result:string* or *undefined*, you can say this as: I want my type to be this or that.

All of these sorts make sense when you think of them as spaces.



TypeScript supports both implicit and explicit types. If you don't write your types explicitly, the compiler will enable type inference to identify the types you're using.

On the other hand, writing them explicitly provides benefits, such as assisting other team members who read your code and verifying that what you see is what the compiler sees.

## **Features of TypeScript**

With the advanced features of TypeScript, it supports additional features of JavaScript, such as object-oriented programming concepts, JS libraries, platform independence, etc.

### ****Compatibility****

TypeScript supports both old and new additional features as well. However, it’s compatible with all the versions of JavaScript, like ES7 and ES12. It can compile the completed code in ES7 back to ES5 and vice versa. This makes sure to have a smooth transition and language portability.

### ****Static Typing****

Static Typing means wherein the developer has to declare the variable type.

Suppose we take variable name str. The code will not run until you give it a type, such as an Integer, Float, List, or anything else. And TypeScript is statically typed.

Static Typing helps you with early bugs detection, faster code completion, and more.

Now, allow us to give you other prominent features here:

* It's easy to maintain and boosts project productivity.
* Feasible to use static typing and annotations
* Supports object-oriented features such as interfaces, inheritance, and classes
* Debugging is simple, and problems are detected at an early stage.
* Supports ES6 (ECMAScript), which provides a simpler syntax for handling objects and inheritance.
* Full-featured IDE support

## **Advantages of TypeScript**

TypeScript offers various advantages which could help you write a better programming language for [**web development solutions**](https://radixweb.com/services/web-development).

### ****Language Features****

Here are other features that TypeScript supports.

* Namespaces
* Interfaces
* Null Checking
* Generics
* Access Modifiers

### ****Optional Parameters****

*// -- TypeScript -- //  
function log(message: string = null) { }  
// -- TypeScript compiled output -- //  
function log(message) {  
if (message === void 0) { message = null; }  
}*

*// -- JavaScript with Babel -- //  
function Log(message = null) { }  
// -- Babel compiled output -- //  
"use strict";  
function Log() {  
var message = arguments.length > 0 && arguments[0] !== undefined ? arguments[0] : null;  
}*

### ****Classes****

*// -- TypeScript -- //  
class Article {  
name: string;  
constructor(name: string) {  
this.name = name;  
}  
}  
// -- TypeScript compiled output -- //  
var Article = /\*\* @class \*/ (function () {  
function Article(name) {  
this.name = name;  
}  
return Article;  
}());*

*// -- JavaScript with Babel -- //  
class Article {  
constructor(name) {  
this.name = name;  
}  
}  
// -- Babel compiled output -- //  
"use strict";  
function \_classCallCheck(instance, Constructor) { if (!(instance instanceof Constructor)) { throw new TypeError("Cannot call a class as a function"); } }  
var Article = function Article(name) {  
\_classCallCheck(this, Article);  
this.name = name;  
};*

## **What’s JavaScript?**

JavaScript – a scripting language, is used to develop interactive web pages. It adheres to the rules of client-side programming; hence, it executes seamlessly in the user’s web browser. However, you can use JavaScript with other technologies like XML and REST APIs.

The primary purpose of JS was to make it a complementary scripting language, similar to how Visual Basic is to C++. However, JavaScript is not suitable for large-level complex applications. It was aimed to write only a few hundred lines of code for an app.

Here are some unique features that JavaScript offers:

* Flexible, dynamic, and cross-platform
* It can be used for both client-side and server-side
* Lightweight interpreted
* All browsers support
* Weakly typed
* JIT compilation

## **Difference Between TypeScript and JavaScript**

While comparing two programming languages – TypeScript and JavaScript, we have to consider many factors, which are as below:

### ****1) Learning Curve****

TypeScript is a superset of JavaScript. To write a TypeScript code, you should have a basic understanding and knowledge of JavaScript. Besides, you should be clear with the OOPS concept as well.

On the other hand, JavaScript is a popular and easy-to-learn scripting language. Many developers use JavaScript with CSS and HTML to create web applications. However, HTML is tough as it contains event handling, web behavior, animations, and scripting.

### ****2) Developers Community****

TypeScript gained its popularity in a short time and was implemented by many enterprises. You will find many tutorials and guides available on the Internet to learn TypeScript. However, it has a very active and supportive community.

On the flip side, JavaScript doesn’t have a large community as compared to TypeScript. JavaScript offers many libraries, frameworks, and code practices. Therefore, it’s advisable to know your [**web development team structure**](https://radixweb.com/blog/guide-to-build-web-development-team-structure) that best suits your business needs for overall team performance.

### ****3) Performance****

As we know, TypeScript was created to overcome the challenges of JavaScript for large-level complex applications. Hence, TypeScript saves development time and allows developers to become more efficient.

The only difference between TypeScript and JavaScript is that TypeScript code is compiled into JavaScript before execution.

### ****4) Syntax****

TypeScript offers variable declaration, functional paradigm, and type system, which JavaScript doesn’t offer. It’s similar to JScript and .Net in terms of syntax, with support for ECMAScript 2015 Standard features including modules, an arrow function syntax, and classes.

JavaScript follows the ECMAScript definition as well. However, it’s not a typed language like TypeScript. It uses many structured programming terminologies from C, like if statements, switch statements, do-while loops, and many more. It offers event-driven style, functional and imperative programming.

### ****5) Tools and Frameworks****

Since Microsoft backs TypeScript, it has many leading frameworks and editors. Through tight integration with editors, it provides error handling during compilation to avoid errors at runtime.

On the flip side, there are many [**JavaScript frameworks**](https://radixweb.com/blog/best-javascript-frameworks-for-web-development) available in the market for web development project requirements. It's a huge ecosystem that's quite popular among programmers. You can quickly find developers who specialize in ReactJS, VueJS, Angular, and other frameworks.

### ****6) Code****

**For TypeScript:**

*class Person  
{  
private name: string;  
constructor (private name: string)  
{  
this.name = name;  
}  
name()  
{  
return “name is “ + this.name;  
}  
}*

**For JavaScript:**

*var Person = (function()  
{  
function Person(personName)  
{  
this.name = personName;  
}  
Person.prototype.name= function()  
{  
return "My name is " + this.name;  
}  
return Person;  
})();*

## **How does TypeScript Differ from JavaScript?**

We know that TypeScript offers more features than JavaScript. TypeScript is a modern-age language – a syntactical superset of JavaScript, whereas JavaScript is a scripting language – a subset of TypeScript.

## **Why Migrate Your Project to TypeScript?**

If you have a large and complex codebase, there are higher chances of errors occurring. However, that would be good if some errors were resolved during the compilation time. This is when you can use TypeScript to reduce errors during compile time. The best part is that the complete Java codebase may be reused as-is.

## **When to Choose: Difference Between JavaScript and TypeScript**

### ****TypeScript****

|  |  |
| --- | --- |
| **Compile Time Type Checking** | With Vanilla JavaScript, type verification is performed at runtime. However, this adds to the runtime overhead, which may be avoided by conducting compile-time validation. |
| **Huge Projects or Multiple Developers** | TypeScript runs seamlessly for large projects or when many developers are working together. |
| **Easy to Work with New Libraries or Frameworks** | Suppose, if you are working with React for the development and are not familiar with its APIs, you can get IntelliSense that will help you identify and navigate new interfaces. However, they both offer type definitions |

### ****JavaScript****

|  |  |
| --- | --- |
| **Small Projects** | TypeScript may be overkill for small projects with fewer codes. |
| **Framework Support** | If TypeScript doesn’t support the choice of your framework – EmberJS, then you may not be able to leverage its features. |
| **Build Tools** | To generate the final JavaScript to be run, TypeScript requires a build step. However, developing JavaScript applications without the use of any build tools is becoming increasingly unusual. |
| **Testing Workflow** | If your talented JavaScript developers are already using test-driven development, the benefits of switching to TypeScript may not be enough to justify the expenditures. |

## **Benefits of TypeScript Over JavaScript**

* TypeScript supports static typing. This means that static typing allows type accuracy to be checked at build time.
* At the time of development, TypeScript identifies the compilation bugs. As a result, the scope of evaluating errors at runtime is very less. JavaScript, on the other hand, is an interpreted language.
* TypeScript is just JS with some extra functionality, namely ES6 features. The TS compiler can compile .ts files into ECMAScript, albeit it may not be kept in your chosen web browser.
* As the codebase grows, TypeScript code is simpler to debug since type errors can be found during compilation rather than during runtime.

## **Will TypeScript Replace JavaScript?**

Well, the shortest answer to the above question is NO!

While talking about TypeScript, it is a completely different language except that it inherits the basic nature of JavaScript. Therefore, JavaScript can’t or will not be replaced ever.

JavaScript is one of the popular and fundamental technologies in the software development industry, used by a plethora of developers for both client-side and server-side.

TypeScript, on the other hand, does not execute in web browsers directly. It transcompiles to JavaScript. JavaScript is easier to debug and compile as it executes directly in the web browser. However, TypeScript is not developed for all types of projects. Therefore, both programming languages have their pros and cons along with a set of characteristics. And to walk along with [**enterprise web development trends**](https://radixweb.com/blog/enterprise-web-development-trends), you have to adopt a new technology or framework for better business processes and customer experience.

## **Head-to-Head Comparison: TypeScript vs JavaScript**

TypeScript and JavaScript share many similarities in developing interactive web pages. While providing the head-to-head comparison of TypeScript vs JavaScript, we can say that JavaScript is a lightweight interpreted and dynamic language which is used for augmenting HTML web pages. On the other hand, TypeScript is an enhanced version of JavaScript. This means TypeScript is a combination of JavaScript and some other traits.

* **TypeScript** is an object-oriented programming language developed by Microsoft Corporation, whereas **JavaScript** is the programming language for the web.
* **TypeScript** comes with many functionalities like code editors and IDEs, whereas **JavaScript** has limited built-in tooling functionality.
* **TypeScript** is an open-source language to build large-scale web apps, whereas **JavaScript** is a server-side programming language that helps to develop interactive web pages.
* **TypeScript** is referred to as an Object-oriented programming language; on the other hand, **JavaScript** is a prototype-based language.
* **TypeScript** offers static types mean variables, objects, and functions, whereas **JavaScript** doesn’t require the explicit declaration of the variables before they’re used.
* **TypeScript** supports multiple interfaces and types to represent data, whereas **JavaScript** doesn’t support such concepts as interfaces.

On the other hand, TypeScript is an enhanced version of JavaScript. This means TypeScript is a combination of JavaScript and some other traits.

**Let’s understand the vital differences in the below comparison table on TypeScript vs JavaScript.**

| **Parameter** | **TypeScript** | **JavaScript** |
| --- | --- | --- |
| **Developed By** | Microsoft in 2012 | Brendan Eich (Netscape) in 1995 |
| **Definition** | TypeScript is a powerful object-oriented language as a superset to JavaScript, with generic and JS features to overcome the complexities of JS. | JavaScript is a scripting language with first-class functions to create dynamic web pages. |
| **Typing** | Strongly Typed. TypeScript supports both static and dynamic typing. | Loosely typed. JavaScript supports only dynamic typing. |
| **Ecosystem** | TypeScript is more of a powerful and intuitive language that supports static typing. | JavaScript is a simple language that optimizes code for compatibility, easy to read and write. |
| **Compilation** | TypeScript needs to be compiled. | JavaScript does not need compilation. |
| **Data Binding** | TypeScript utilizes concepts like interfaces and types to define the data being employed. | No such notion is presented in JavaScript. |
| **Learning Curve** | TypeScript has a stiff learning curve. Also, it needs scripting knowledge. | It is easy to learn and a flexible language for writing web scripts. |
| **Npm Packages** | With Typescript, numerous npm packages either have static type definitions or have an exterior one that is simpler to install. | JavaScript provides the alternative to search and form code without any build step. |
| **Client-Side or Server-Side** | TypeScript is specially used on the client-side. | JavaScript is used on both server-side and client-side. |
| **Files Extensions** | .tsx and .ts | .js |
| **Community** | TypeScript has a smaller community of software developers. | JavaScript has a large community of software developers. |
| **Prototyping** | Prototyping feature is available in TypeScript. | JS doesn’t support prototyping. |
| **Companies and Websites** | Asana, Clever, Screen award | Airbnb, Codecademy, Instagram |
| **Code Example** | function multiply (a, b){ return a\*b;}var result = multiply(a, b);console.log(‘The answer is - ’ + result); | *<script>function multiply (a, b){ return a\\*b;}var result = multiply(a, b);document.write (‘The answer is – ’ + result);</script>* |
|  |  |  |

# **TypeScript Arrays**

TypeScript has a specific syntax for typing arrays.

const names: string[] = [];  
names.push("Dylan"); // no error  
// names.push(3); // Error: Argument of type 'number' is not assignable to parameter of type 'string'.

## **Readonly**

The readonly keyword can prevent arrays from being changed.

### Example

const names: readonly string[] = ["Dylan"];  
names.push("Jack"); // Error: Property 'push' does not exist on type 'readonly string[]'.  
// try removing the readonly modifier and see if it works?

## **Type Inference**

TypeScript can infer the type of an array if it has values.

### Example

const numbers = [1, 2, 3]; // inferred to type number[]  
numbers.push(4); // no error  
// comment line below out to see the successful assignment  
numbers.push("2"); // Error: Argument of type 'string' is not assignable to parameter of type 'number'.  
let head: number = numbers[0]; // no error

# **TypeScript Functions**

TypeScript has a specific syntax for typing function parameters and return values.

## **Return Type**

The type of the value returned by the function can be explicitly defined.

### Example

// the `: number` here specifies that this function returns a number  
function getTime(): number {  
  return new Date().getTime();  
}

If no return type is defined, TypeScript will attempt to infer it through the types of the variables or expressions returned.

## **Void Return Type**

The type void can be used to indicate a function doesn't return any value.

### Example

function printHello(): void {  
  console.log('Hello!');  
}

## **Parameters**

Function parameters are typed with a similar syntax as variable declarations.

### Example

function multiply(a: number, b: number) {  
  return a \* b;  
}

If no parameter type is defined, TypeScript will default to using any

## **Optional Parameters**

By default TypeScript will assume all parameters are required, but they can be explicitly marked as optional.

### Example

// the `?` operator here marks parameter `c` as optional  
function add(a: number, b: number, c?: number) {  
  return a + b + (c || 0);  
}

## **Default Parameters**

For parameters with default values, the default value goes after the type annotation:

### Example

function pow(value: number, exponent: number = 10) {  
  return value \*\* exponent;  
}

TypeScript can also infer the type from the default value.

## **Named Parameters**

Typing named parameters follows the same pattern as typing normal parameters.

### Example

function divide({ dividend, divisor }: { dividend: number, divisor: number }) {  
  return dividend / divisor;  
}

## **Rest Parameters**

Rest parameters can be typed like normal parameters, but the type must be an array as rest parameters are always arrays.

### Example

function add(a: number, b: number, ...rest: number[]) {  
  return a + b + rest.reduce((p, c) => p + c, 0);  
}

## **Type Alias**

Function types can be specified separately from functions with type aliases.

Example

type Negate = (value: number) => number;  
  
// in this function, the parameter `value` automatically gets assigned the type `number` from the type `Negate`  
const negateFunction: Negate = (value) => value \* -1;

# **TypeScript Classes**

TypeScript adds types and visibility modifiers to JavaScript classes.

## **Members: Types**

The members of a class (properties & methods) are typed using type annotations, similar to variables.

### Example

class Person {  
  name: string;  
}  
  
const person = new Person();  
person.name = "Jane";

## **Members: Visibility**

Class members also be given special modifiers which affect visibility.

There are three main visibility modifiers in TypeScript.

* public - (default) allows access to the class member from anywhere
* private - only allows access to the class member from within the class
* protected - allows access to the class member from itself and any classes that inherit it
* Example
* class Person {  
    private name: string;  
    
    public constructor(name: string) {  
      this.name = name;  
    }  
    
    public getName(): string {  
      return this.name;  
    }  
  }  
    
  const person = new Person("Jane");  
  console.log(person.getName()); // person.name isn't accessible from outside the class since it's private

## **Parameter Properties**

TypeScript provides a convenient way to define class members in the constructor, by adding a visibility modifiers to the parameter.

Example

class Person {  
  // name is a private member variable  
  public constructor(private name: string) {}  
  
  public getName(): string {  
    return this.name;  
  }  
}  
  
const person = new Person("Jane");  
console.log(person.getName());

## **Readonly**

Similar to arrays, the readonly keyword can prevent class members from being changed.

### Example

class Person {  
  private readonly name: string;  
  
  public constructor(name: string) {  
    // name cannot be changed after this initial definition, which has to be either at it's declaration or in the constructor.  
    this.name = name;  
  }  
  
  public getName(): string {  
    return this.name;  
  }  
}  
  
const person = new Person("Jane");  
console.log(person.getName());

## **Inheritance: Implements**

Interfaces can be used to define the type a class must follow through the implements keyword.

### Example

interface Shape {  
  getArea: () => number;  
}  
  
class Rectangle implements Shape {  
  public constructor(protected readonly width: number, protected readonly height: number) {}  
  
  public getArea(): number {  
    return this.width \* this.height;  
  }  
}

A class can implement multiple interfaces by listing each one after implements, separated by a comma like so: class Rectangle implements Shape, Colored {

## **Inheritance: Extends**

Classes can extend each other through the extends keyword. A class can only extends one other class.

### Example

interface Shape {  
  getArea: () => number;  
}  
  
class Rectangle implements Shape {  
  public constructor(protected readonly width: number, protected readonly height: number) {}  
  
  public getArea(): number {  
    return this.width \* this.height;  
  }  
}  
  
class Square extends Rectangle {  
  public constructor(width: number) {  
    super(width, width);  
  }  
  
  // getArea gets inherited from Rectangle  
}

## **Override**

When a class extends another class, it can replace the members of the parent class with the same name.

Newer versions of TypeScript allow explicitly marking this with the override keyword.

### Example

interface Shape {  
  getArea: () => number;  
}  
  
class Rectangle implements Shape {  
  // using protected for these members allows access from classes that extend from this class, such as Square  
  public constructor(protected readonly width: number, protected readonly height: number) {}  
  
  public getArea(): number {  
    return this.width \* this.height;  
  }  
  
  public toString(): string {  
    return `Rectangle[width=${this.width}, height=${this.height}]`;  
  }  
}  
  
class Square extends Rectangle {  
  public constructor(width: number) {  
    super(width, width);  
  }  
  
  // this toString replaces the toString from Rectangle  
  public override toString(): string {  
    return `Square[width=${this.width}]`;  
  }  
}

By default the override keyword is optional when overriding a method, and only helps to prevent accidentally overriding a method that does not exist. Use the setting noImplicitOverride to force it to be used when overriding.

## **Abstract Classes**

Classes can be written in a way that allows them to be used as a base class for other classes without having to implement all the members. This is done by using the abstract keyword. Members that are left unimplemented also use the abstract keyword.

### Example

abstract class Polygon {  
  public abstract getArea(): number;  
  
  public toString(): string {  
    return `Polygon[area=${this.getArea()}]`;  
  }  
}  
  
class Rectangle extends Polygon {  
  public constructor(protected readonly width: number, protected readonly height: number) {  
    super();  
  }  
  
  public getArea(): number {  
    return this.width \* this.height;  
  }  
}

Abstract classes cannot be directly instantiated, as they do not have all their members implemented.

Angular components

Components are the main building blocks for Angular applications. Each component consists of:

* An HTML template that declares what renders on the page
* A TypeScript class that defines behavior
* A CSS selector that defines how the component is used in a template
* Optionally, CSS styles applied to the template

## **Creating a component**

The best way to create a component is with the Angular CLI. You can also create a component manually.

### Creating a component using the Angular CLI

To create a component using the Angular CLI:

1. From a terminal window, navigate to the directory containing your application.
2. Run the ng generate component <component-name> command, where <component-name> is the name of your new component.

By default, this command creates the following:

* A directory named after the component
* A component file, <component-name>.component.ts
* A template file, <component-name>.component.html
* A CSS file, <component-name>.component.css
* A testing specification file, <component-name>.component.spec.ts

Where <component-name> is the name of your component.

### Creating a component manually

Although the Angular CLI is the best way to create an Angular component, you can also create a component manually.

To create a new component manually:

1. Navigate to your Angular project directory.
2. Create a new file, <component-name>.component.ts.
3. At the top of the file, add the following import statement.

import { [Component](https://angular.io/api/core/Component) } from '@angular/core';

1. After the import statement, add a @[Component](https://angular.io/api/core/Component) decorator.

@[Component](https://angular.io/api/core/Component)({

})

1. Choose a CSS selector for the component.

@[Component](https://angular.io/api/core/Component)({

selector: 'app-component-overview',

})

1. Define the HTML template that the component uses to display information. In most cases, this template is a separate HTML file.

@[Component](https://angular.io/api/core/Component)({

selector: 'app-component-overview',

templateUrl: './component-overview.component.html',

})

1. Select the styles for the component's template. In most cases, you define the styles for your component's template in a separate file.

@[Component](https://angular.io/api/core/Component)({

selector: 'app-component-overview',

templateUrl: './component-overview.component.html',

styleUrls: ['./component-overview.component.css']

})

1. Add a class statement that includes the code for the component.

export class ComponentOverviewComponent {

}

## **Defining a component's template**

A template is a block of HTML that tells Angular how to render the component in your application. Define a template for your component in one of two ways: by referencing an external file, or directly within the component.

To define a template as an external file, add a templateUrl property to the @[Component](https://angular.io/api/core/Component) decorator.

@[Component](https://angular.io/api/core/Component)({

selector: 'app-component-overview',

templateUrl: './component-overview.component.html',

})

To define a template within the component, add a template property to the @[Component](https://angular.io/api/core/Component) decorator that contains the HTML you want to use.

@[Component](https://angular.io/api/core/Component)({

selector: 'app-component-overview',

template: '<h1>Hello World!</h1>',

})

If you want your template to span multiple lines, use backticks (`). For example:

@[Component](https://angular.io/api/core/Component)({

selector: 'app-component-overview',

template: `

<h1>Hello World!</h1>

<p>This template definition spans [multiple](https://angular.io/api/forms/SelectMultipleControlValueAccessor) lines.</p>

`

})

An Angular component requires a template defined using template or templateUrl. You cannot have both properties in a component.

## **Declaring a component's styles**

Declare component styles used for its template in one of two ways: By referencing an external file, or directly within the component.

To declare the styles for a component in a separate file, add a styleUrls property to the @[Component](https://angular.io/api/core/Component) decorator.

@[Component](https://angular.io/api/core/Component)({

selector: 'app-component-overview',

templateUrl: './component-overview.component.html',

styleUrls: ['./component-overview.component.css']

})

To declare the styles within the component, add a styles property to the @[Component](https://angular.io/api/core/Component) decorator that contains the styles you want to use.

@[Component](https://angular.io/api/core/Component)({

selector: 'app-component-overview',

template: '<h1>Hello World!</h1>',

styles: ['h1 { font-weight: normal; }']

})

The styles property takes an array of strings that contain the CSS rule declarations.

**Case Study: Task Management System**

Imagine you are tasked with building a simple Task Management System using TypeScript. The system should allow users to add, remove, and list tasks. Each task has a title, description, and a status (e.g., "Incomplete" or "Complete"). Your goal is to design the data structure and operations for managing tasks efficiently.

**Requirements:**

1. **Task Model:**
   * Define a TypeScript interface or type to represent the structure of a task, including properties such as title, description, and status.
2. **Array Initialization:**
   * Declare and initialize an array of tasks in TypeScript when the application starts.
3. **Add Task Functionality:**
   * Implement a function that adds a new task to the array. The function should take a title and description as parameters and set the status to "Incomplete" by default.
4. **List Tasks Functionality:**
   * Implement a function that lists all tasks in the array. Display each task's title, description, and status.
5. **Remove Task Functionality:**
   * Implement a function that removes a task from the array based on its title.
6. **Update Task Status Functionality:**
   * Implement a function that updates the status of a task to "Complete" when the task is marked as complete.

**Considerations:**

* Ensure that the TypeScript type system is leveraged to enforce proper data structures.
* Implement functions with clear input and output types.
* Handle edge cases, such as trying to remove a task that doesn't exist.

**Questions:**

1. How would you design the TypeScript type/interface for the task structure?
2. Demonstrate how you would initialize an array of tasks at the beginning of the application.
3. Write a TypeScript function that adds a new task to the array and explain your choice of data structures.
4. Implement a TypeScript function to list all tasks in the array. How would you handle an empty task list?
5. Provide a TypeScript function to remove a task from the array based on its title. How would you handle a non-existent task?
6. Explain how you would implement a function to update the status of a task to "Complete" in a type-safe manner.

How do you declare a function in TypeScript?

Can you have a mix of default and non-default parameters in a TypeScript function?

What are rest parameters, and how are they declared in TypeScript?

Explain the concept of function overloading in TypeScript.