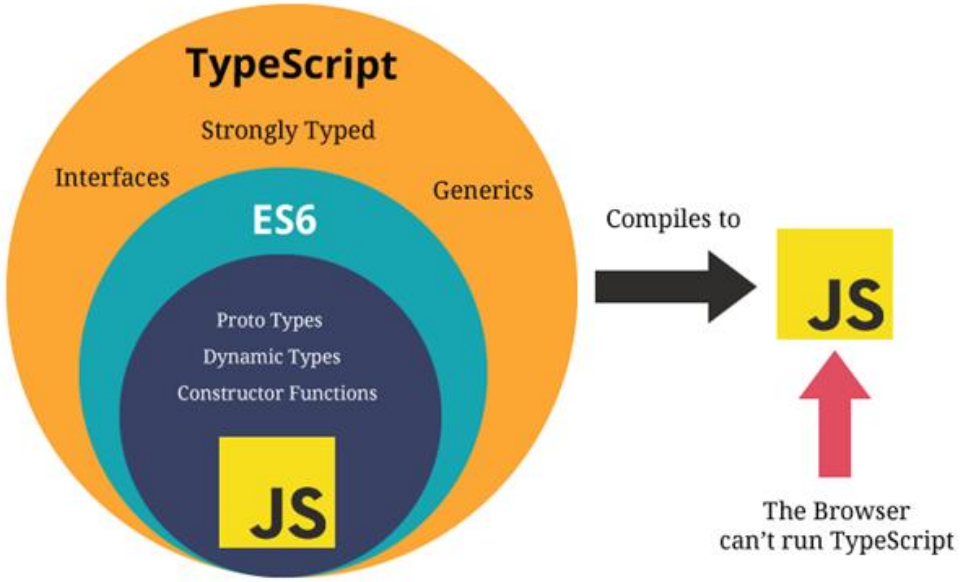
CS472 WAP TypeScript

Maharishi International University - Fairfield, Iowa

JavaScript vs. TypeScript

JavaScript is the most popular programming language on the web. It is used to create websites, servers, games, and native mobile apps. JavaScript is a loosely typed language.

TypeScript is a free and open-source high-level programming language developed by Microsoft that adds static typing with optional type annotations to JavaScript. It is designed for the development of large applications and transpiles to JavaScript.



Why TypeScript?

One of the great things about type-checking is that:

1. It helps write safe code because it can **prevent bugs at compile time**.
2. Compilers can improve and run the code **faster**.

Types are **optional** in TypeScript *(because they can be inferred)*.

TypeScript Compiler

TypeScript compiles into simple JavaScript.

A TypeScript code is written in a file with `.ts` extension and then compiled into JavaScript using the TypeScript compiler.

Note: if you desire to transpile the ts files to js, you may use the following command: `npx tsc`

TypeScript Development Workspace

```
// install Node
// within a blank folder, create package.json
npm init

// install the following development dependencies
npm i tsx @types/node -D

// create .gitignore and exclude node_modules from being controlled by Git

// update package.json file with a new script
{"start": "tsx watch filename.ts"}

// run your TS script
npm run start
```

Type Annotations

We can specify the type using **`:type`** after the name of the variable, parameter or property.

TypeScript includes all the primitive types of JavaScript- number, string and boolean.

```
const grade: number = 90; // number variable
const name: string = "Asaad"; // string variable
const isFun: boolean = true; // Boolean variable
```

Type Inference

It is not mandatory to annotate types in TypeScript, as it infers types of variables when there is no explicit information available in the form of type annotations.

```
let text = "some text";
text = 123; // Type '123' is not assignable to type 'string'
```

any

Never ever use the type **any** especially to silence an error or when you do not have prior knowledge about the type of some variables.

```
let something: any = 'Asaad';
something = 569;
something = true;

let data: any[] = ["Asaad", 569, true];
```

enum

Enums allow us to declare **a set of named Constants**, a collection of related values that can be numeric or string values.

Enum values start from zero and increment by 1 for each member.

Enum in TypeScript supports **reverse mapping**.

```
enum Technologies {
  Angular,
  React,
  ReactNative
}

// Technologies.React; returns 1
// Technologies["React"]; returns 1
// Technologies[0]; returns Angular
```

```
console.log(Technologies);

{
    '0': 'Angular',
    '1': 'React',
    '2', 'ReactNative',
    Angular: 0,
    React: 1,
    ReactNative: 2
}
```

Union Type

Union type allows us to combine more than one data type.

```
(type1 | type2 | type3 | .. | typeN)
```

```
let course: (string | number);
let data: string | number;
```

Explicit Array Type

There are two ways to declare an array type:

1. Using **square brackets**

```
let values: number[] = [12, 24, 48];
```

2. Using a **generic array type**, `Array<elementType>`

```
let fruits: Array<string> = ['Apple', 'Orange', 'Banana'];
```

You can always initialize an array with many data types elements, but you will not get the advantage of TypeScript's type system.

Type Assertion

1. Using the angular bracket <> syntax

```
let code: any = 123;
let courseCode = <number> code;
```

2. Using as keyword

```
let code: any = 123;
let courseCode = code as number;
```

Note: The assertion from type S to T succeeds if either S is a subtype of T or T is a subtype of S. Asserting to unknown (or any) is compatible with all types.

Object Types

We may represent object types as an **interface** or a **type alias**:

```
interface Person {
  name: string;
  age: number;
}
```

OR

```
type Person = {
  name: string;
  age: number;
};
```

```
function greet(person: Person) {
  return "Hello " + person.name;
}
```

Combine Types

```
type Person = {
    name: string;
    age: number;
};

type Tech = {
    phone: string;
    laptop: string;
};

type TechPerson = Person & Tech;

const anna: TechPerson = { name: “Anna", age: 20, phone: "Pixel", laptop: "MacBook Pro" };
```

Combine Interfaces

```
interface Person {
  name: string;
  age: number;
}

interface Tech {
  phone: string;
  laptop: string;
}

const anna: Person & Tech = { name: “Anna", age: 0, phone: "Pixel", laptop: "MacBook Pro" };
```

Utility Types

```
interface Person {
   firstname: string;
   lastname: string;
   age: number;
   height: number;
   weight: number;
}

type NamedPerson = Pick<Person, "firstname" | "lastname">;
const anna: NamedPerson = { firstname: "Anna", lastname: "Smith" };

type SomePerson = Partial<Person>;
const mike: SomePerson = { firstname: "Mike", height: 180 };

type JustName = Omit<Person, "age" | "height" | "weight">;
const smith: JustName = { firstname: "Smith", lastname: "John" };
```