Slide Note

Learning Consolidation

Build Interactive Web Pages With TypeScript

Slide Note

In this sprint, you learned to:

- Explain the need for TypeScript language
- Identify the different data types available in TypeScript
- Execute functions with type annotations
- Define object types using type aliases
- Create classes to build multiple objects
- Build a simple interactive web page using TypeScript

Slide Note

What Is TypeScript?

- TypeScript is a strongly-typed programming language developed as a superset of JavaScript.
- TypeScript allows the programmer to define types to increase the reliability of a program by avoiding type mismatches.
- Datatypes are assigned by simply placing a colon after the variable name but before the equal sign:

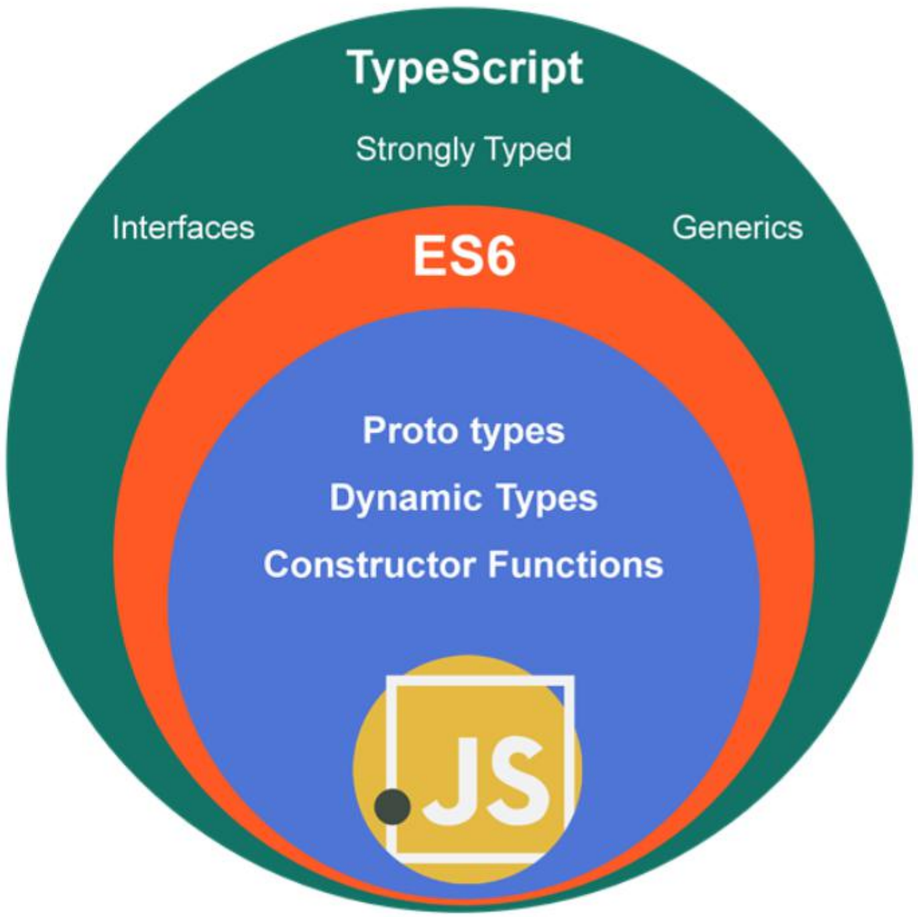
  Let {variable name}: {variable type} = {variable value}

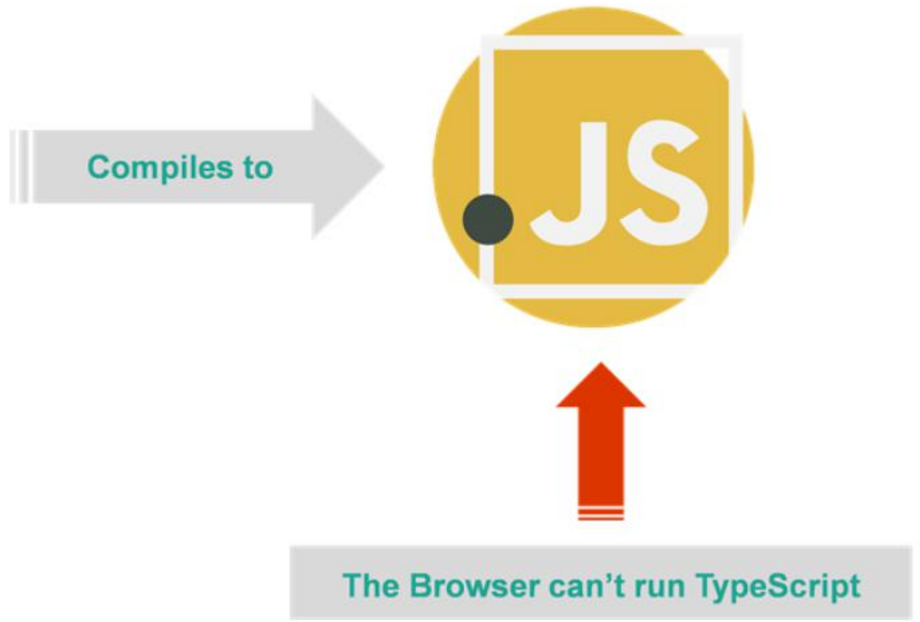
- TypeScript files are saved with the .ts extension.

```
//declare a variable with data type
let personName: string = "John";
console.log("Hello " + personName);

//declare 2 variables with
numeric type
let num1: number = 10;
let num2: number = 20;
console.log(`Product of 2
numbers: ${num1 * num2}`);

//declare a variable with boolean
type
let isCitizen: boolean = true;
let age: number= 23;
if(isCitizen && age > 18){
console.log("Eligible for voting");
}
```

Slide Note

Typescript: Superset of JavaScript





Slide Note

For running a TypeScript code, there are two ways:

- After compiling the TypeScript file to JavaScript, add .js file in index.html
- Run the JavaScript file in the terminal by using the command **node .js**
- Differences between compiler and interpreter
  - https://www.programiz.com/article/difference-compiler-interpreter
  - https://softwareengineeringdaily.com/2018/10/03/javascript-and-the-inner-workings-of-your-browser/

TypeScript Code Execution

Steps to Execute TypeScript Code

1. Install TypeScript as an NPM package on your local machine or in your project by using:
   npm install –g typescript

2. Check the installed version of typescript by using:
   tsc –v

3. Compile the file by using:
   tsc <filename>.ts

4. Include the <filename>.js file generated in index.html to see the output in the browser's developer console

Note: Steps 1 and 2 are done once for the TypeScript installation. Steps 3 and Step 4 should be followed for running any TypeScript file.

```
function add(num1, num2) {
return num1 + num2;
}
//declare variables with number datatype
let a: number = 10;
let b: number = 20;
//Executes and returns the sum
console.log("Sum " + add(a, b));
//declare variable with string data type
let c: string = "John";
//Gives Compilation error as argument
type is not matching
console.log("Sum "+ add(a,c));
```

Slide Note

Statically vs. Dynamically-Typed Languages

Statically-Typed Languages	Dynamically-Typed Languages
A language is statically-typed if the type of a variable is known at compile time.	A language is dynamically-typed if the type is associated with run-time values, and not named variables or fields.
The programmer must specify the type of each variable.	The programmer can write quicker because there is no need to specify the type every time.
The advantage here is that all kinds of checking can be done by the compiler, and therefore a lot of trivial bugs are caught at a very early stage.	There is no compiler to do static type checking and there are chances of the interpreter misinterpreting the type of a variable.
Languages: C, C++, Java, Rust, Go, TypeScript	Languages: Perl, Ruby, Python, PHP, JavaScript
Example: `String foo = “Foo”;`	Example: `let foo = "Foo";`

Slide Note

Different Data Types in TypeScript

Data Type	Description	Example
string	Textual data	let str: string = "hello";
number	An integer or a floating-point number	let value: number = 100;
boolean	Any of the two values: true or false	let hasText: boolean = true;
enum	Declares a set of named constants	enum level { Low, MEDIUM, HIGH}
union	Type formed from two or more other types	let code: (string | number);
any	Data type used when type checking errors need to be ignored	let value: any = "hello"; let value: any = 1234;
array	A sequential storage of data accessed by index	let marks: number[]= {55,89};
tuple	Similar to an array but elements are of different data types	let person: [string, number] = ["Kunal", 2018];

Slide Note

TypeScript Functions

- TypeScript functions allow us to add type annotations after each parameter to declare the type of parameter that the function accepts.
  - **Parameter type annotations**: Allows the ability to add type annotations after each parameter to declare the type of parameter that the function accepts
  - **Return type annotations**: Specifies the type that the function returns and appears after the parameter list

- Anonymous functions are functions without names. When type annotations are not specified in anonymous functions, TypeScript can understand the type based on the context.

- Arrow functions with type annotations are also used in TypeScript.

Slide Note

Object Types and Classes in TypeScript

- Class is basically a blueprint to create a specific type of objects whose state and behavior is represented by the variables and methods defined inside the class.
- A class definition can contain the following properties:
  - **Field:** It is a variable declared in a class.
  - **Method:** It represents an action for the object.
  - **Constructor**: It is responsible for initializing the object in memory.
- In TypeScript, objects are represented through object type. Object types are defined by listing their properties and the types of each property.

Slide Note

Access specifiers are not available in JavaScript code, so all class members are accessible outside the class to modify it. According to the encapsulation concept, the visibility of data members should be restricted and should be available to modify outside the class body only if it is needed.

TypeScript supports these access specifiers. As a result, you can restrict the modification of data members by declaring them as private. The above JS code allows access and modifies the studentID outside the class.

But in the TS code, declaring studentID as private restricts its access so it can not modified once it is constructed.

JavaScript Code vs. TypeScript Code

```
class Student {
constructor(studentID, studentName) {
this.studentID = studentID;
this.studentID = studentName;
}
}

let st = new Student(111, "John");
//Can access the studentID value outside
st.studentID = 123;
st.studentName = "John Kevin";

console.log("Student id:" + st.studentID);
```

```
class Student {
private studentID: number;
studentName: string;
constructor(studentID: number, studentName:
string) {
this.studentID=studentID;
this.studentName=studentName;
}
getStudentId():number{return this.studentID;}
};
let ob = new Student(111,"John");

 //compilation error-cannot access outside the
class definition once created
ob.studentID = 123;
ob.studentName="John Kevin";
//properties are accessed through methods
console.log("Student id:" + ob.getStudentId())
```

Slide Note

TypeScript Class Members

- In object-oriented programming, class members can be made public or private, i.e., a class can control the visibility of its data members. This is done using access modifiers.
- There are three types of access modifiers in TypeScript:
  - private
  - public
  - protected
- Class members are public by default.
- TypeScript classes also have getters/setters.
- Private property names are prefixed with _(underscore).

Slide Note

Typescript Configuration File

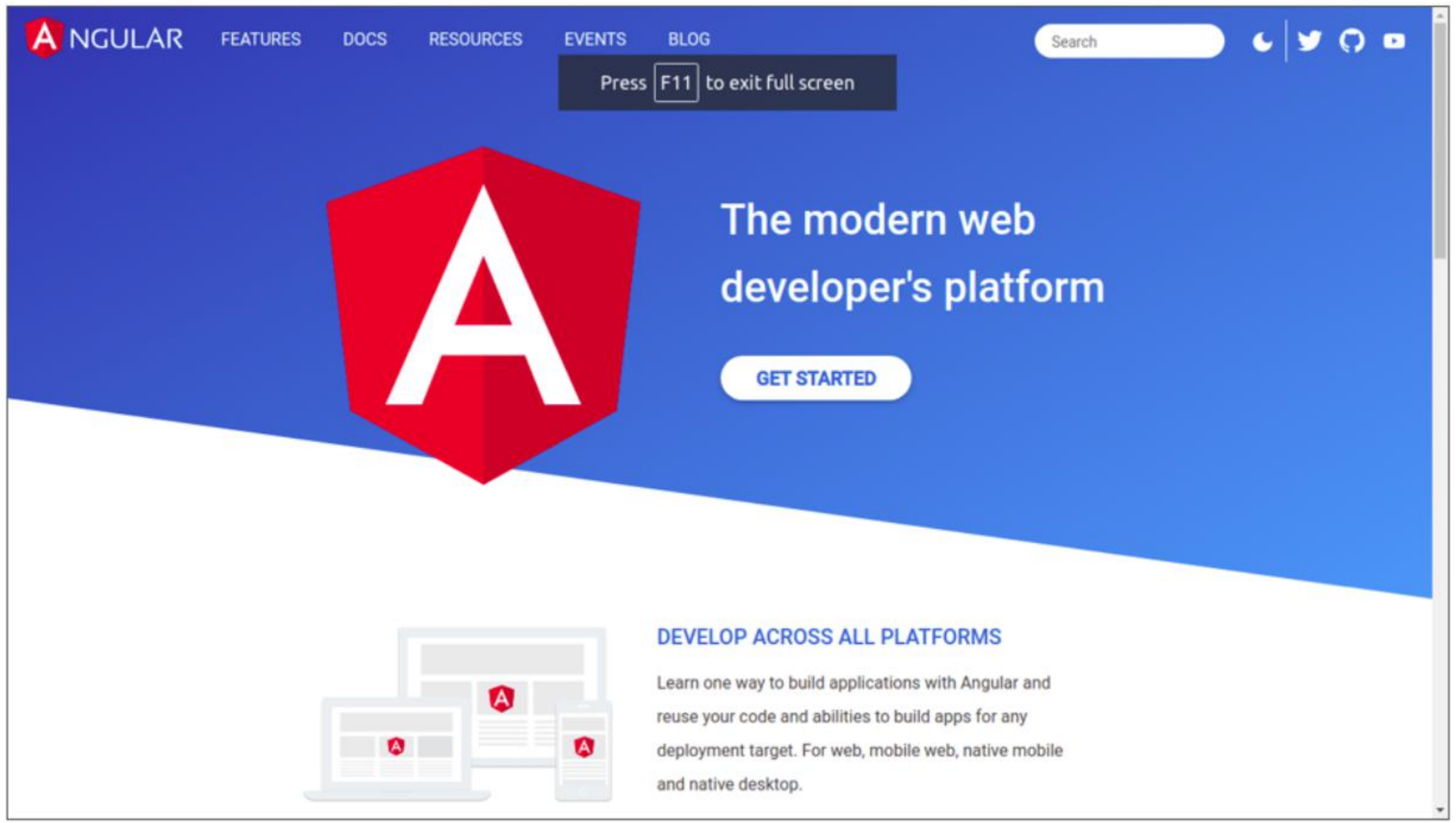
- While creating a TypeScript project, **tsconfig.json** file is used which specifies the root files and the compiler options required to compile the project.
- The presence of a **tsconfig.json** file in a directory indicates that the directory is the root of a TypeScript project.
- Compiler Options: These options specifies the TypeScript's configuration and covers how the language should work.
- Include: Specifies an array of file names or patterns to include in the program. These file names are resolved relative to the directory containing the **tsconfig.json** file.

tsconfig.json

```
{
"compilerOptions":
{
   "target": "es2019"/* Specify
ECMAScript target version */,
   "module": "commonjs",
   "outDir": "public/js" /*Redirect
output structure to the directory*/,
   "rootDir": "src" /* Specify the root
directory of input files. */,
  "strict": true /* Enable all strict
type-checking options. */,
},
  "include": ["src"]
}
```

Slide Note

The Angular Website When Migrated From JavaScript to TypeScript



Slide Note

JavaScript to TypeScript Converter

- A tool called ts-migrate will automatically convert the whole JavaScript project to a compiling TypeScript.
- It won't generate the perfect type-safe code, but it's a good starting point that will significantly speed up the migration process.
- There will be @ts-expect-error and any all over the place that you'll have to fix manually over time.
- It will create tsconfig.json for you with a reasonable default configuration.
- It will change the file extensions for all the source code files from .js to .ts.

Slide Note

Self-Check

Which of the following is a valid tuple?

1. `let emp: [number, string] = [1, "James"]`
2. `let emp: (number | string) = 555`
3. `let emp: string[ ] = ['Sam'];`
4. None of the above

Slide Note

Self-Check: Solution

Which of the following is a valid tuple?

1. `let emp: [number, string] = [1, "James"]`
2. `let emp: (number | string) = 555`
3. `let emp: string[ ] = ['Sam'];`
4. None of the above

Explanation:
Option 2: Union type declaration
Option 3: Array type declaration

Slide Note

Self-Check

Which of the following is the valid arrow function?

```
1. let sum = (x: number, y: number) => x+y;
2. let sum() = (x: number, y: number): number => x+y;
3. let sum = (x: number, y: number) => return x+y;
4. let sum: (x: number, y: number) => x+y;
```

Slide Note

Self-Check: Solution

Which of the following is the valid arrow function?

1. `let sum = (x: number, y: number) => x+y;`
2. `let sum() = (x: number, y: number): number => x+y;`
3. `let sum = (x: number, y: number) => return x+y;`
4. `let sum: (x: number, y: number) => x+y;`