WEB ENGINEERING

PRESENTATION TYPESCRIPT

PRESENTED TO:
DR. LALIT PUROHIT SIR
MR. UPENDRA SIR

PRESENTED BY:
HARIOM GAUTAM - IT31
HIMANSHU GUPTA - IT 36
PAVESH KANUNGO - IT 58

What is **TYPESCRIPT?**

- Free and open-source high-level programming language
- Launched in 2012
- Developed and maintained by Microsoft
- Superset of JavaScript
- The main thing TypeScript provides is static typing

Key Features TYPESCRIPT

1. **Static Typing:** allowing developers to specify the types of variables, function parameters, and return values. This helps catch type-related errors during development rather than at runtime.
2. **Object-Oriented Programming (OOP):** TypeScript supports object-oriented programming concepts such as classes, interfaces, inheritance, and encapsulation. This makes it more suitable for large-scale, maintainable codebases.

Key Features TYPESCRIPT

3. ES6 and Beyond Support: TypeScript supports features from ECMAScript 6 (ES6) and later versions of JavaScript. This includes arrow functions, classes, modules, and more, providing developers with modern language features.

4. Tooling Support: TypeScript comes with a compiler (tsc) that translates TypeScript code into JavaScript. Integrated Development Environments (IDEs) such as Visual Studio Code have built-in support for TypeScript, offering features like autocompletion, error checking, and refactoring tools.

Key Features TYPESCRIPT

5. **Compatibility with Existing JavaScript Code:** Since TypeScript is a superset of JavaScript, existing JavaScript code can be gradually migrated to TypeScript. Developers can start by adding type annotations to portions of their code and incrementally adopt TypeScript features.

6. **Community and Ecosystem:** TypeScript has gained popularity in the developer community, and many open-source projects and libraries are written in TypeScript. It has a strong ecosystem and is commonly used in web development, particularly in combination with popular frontend frameworks like Angular.

Types

In a programming language, types refer to the kind or type of information a given program stores.

In JavaScript, there are six basic data types which can be divided into three main categories:

1

PRIMITIVE DATA TYPES

STRING, NUMBER, AND BOOLEAN

2

COMPOSITE DATA TYPES

OBJECT, ARRAY, AND FUNCTION

3

SPECIAL DATA TYPES

UNDEFINED AND NULL

About Types

Primitive Data types

Strings: represents textual data

```
let a = "Hi there!";
```

Numbers: positive or negative numbers with or without decimal place

```
let a = 25;
```

Boolean: 2 values, true and false

```
let areYouEnjoyingTheArticle = true;
```

About Types

Composite Data types

Objects: stores collection of data as key-value pair

```
let car = {
  modal: "BMW X3",
  color: "white",
  doors: 5
};
```

Arrays: stores multiple values in a single variable

```
let arr = ["I", "love", "freeCodeCamp"];

console.log(arr[2]); // Output: freeCodeCamp
```

Functions: object that executes a block of code

```
let greeting = function () {
  return "Hello World!";
};

console.log(greeting()); // Output: Hello World!
```

About Types

Special Data types

Undefined: variable declared, but no value assigned

```
let a;

console.log(a); // Output: undefined
```

Null: no value

```
let thisIsEmpty = null;
```

WHAT'S THE DEAL WITH TYPES AND JAVASCRIPT?

Why something like TypeScript?

JavaScript is a loosely typed and dynamic language. This means that, variables are not directly associated with any particular value type, and any variable can be assigned (and re-assigned) values of all types.

```
let foo = 42; // foo is now a number
foo = "bar";  // foo is now a string
foo = true;   // foo is now a boolean
```

when building huge applications, dynamic types can lead to silly bugs in the code base.

```
const personDescription = (name, city, age) =>
  `${name} lives in ${city}. he's ${age}. In 10 years he'll be ${age + 10}`;
```

```
console.log(personDescription("Germán", "Buenos Aires", 29));
// Output: Germán lives in Buenos Aires. he's 29. In 10 years he'll be 39.
```

But if accidentally we pass the function the third parameter as a string, we get a wrong output

```
console.log(personDescription("Germán", "Buenos Aires", "29"));
// output: Germán lives in Buenos Aires. he's 29. In 10 years he'll be **2910**.
```

JavaScript doesn't show an error because the program doesn't have a way of knowing what type of data the function should receive.

IN COMES TYPESCRIPT

- In TypeScript, much like in other programming languages such as Java or C#, we need to declare a data type whenever we create a data structure.
- If there's a match, the program runs, and if not, we get an error and these errors are very valuable, because as developers we can catch bugs earlier.

In JavaScript

```
const personDescription = (name, city, age) =>
  `${name} lives in ${city}. he's ${age}. In 10 years he'll be ${age + 10}`;
```

In TypeScript

```
const personDescription = (name: string, city: string, age: number) =>
  `${name} lives in ${city}. he's ${age}. In 10 years he'll be ${age + 10}.`;
```

Calling the function with the wrong parameter data type

```
console.log(personDescription("Germán", "Buenos Aires", "29"));
```

// Error: TSError: × Unable to compile TypeScript: Argument of type 'string' is not assignable to parameter of type 'number'.

TYPESCRIPT

Basics

Typescript Basics

Types by Inference

In type by **inference**, you don't declare a type at all, but TypeScript infers (guesses) it for you.

```
let helloWorld = "Hello World";
```

If we now try to reassign it to a number, we'll get the following error:

```
helloWorld = 20;
// Type 'number' is not assignable to type 'string'.ts(2322)
```

Typescript Basics

Declaring Types

The syntax to declare types is quite simple: we just add a colon and its type to the right of whatever we're declaring.

```
let myName: string = "Germán";
```

If we try to reassign it to a number, we'll get the following error:

```
myName = 36; // Error: Type 'number' is not assignable to type 'string'.
```

Typescript Basics

Interfaces

- An interface looks a lot like a JavaScript object – but we use the interface keyword, we don't have an equal sign or commas, and besides each key we have its data type instead of its value.

```
interface myData {
  name: string;
  city: string;
  age: number;
}

let myData: myData = {
  name: "Germán",
  city: "Buenos Aires",
  age: 29
};
```

Typescript Basics

Say again I pass the age as a string, I'll get the following error:

```
let myData: myData = {
  name: "Germán",
  city: "Buenos Aires",
  age: "29" // Output: Type 'string' is not assignable to type 'number'.
};
```

Typescript Basics

Conditionals

If we wanted to make a key conditional, allowing it to be present or not, we just need to add a question mark at the end of the key in the interface:

```
interface myData {
  name: string;
  city: string;
  age?: number;
}
```

Typescript Basics

Unions

If we want a variable to be able to be assigned more than one different data type, we can declare so by using **unions** like this:

```
interface myData {
  name: string;
  city: string;
  age: number | string;
}

let myData: myData = {
  name: "Germán",
  city: "Buenos Aires",
  age: "29" // I get no error now
};
```

Typescript Basics

Typing Functions

When typing functions, we can type its parameters as well as its return value:

```
interface myData {
  name: string;
  city: string;
  age: number;
  printMsg: (message: string) => string;
}

let myData: myData = {
  name: "Germán",
  city: "Buenos Aires",
  age: 29,
  printMsg: (message) => message
};

console.log(myData.printMsg("Hola!"));
```

Typescript Basics

Typing Arrays

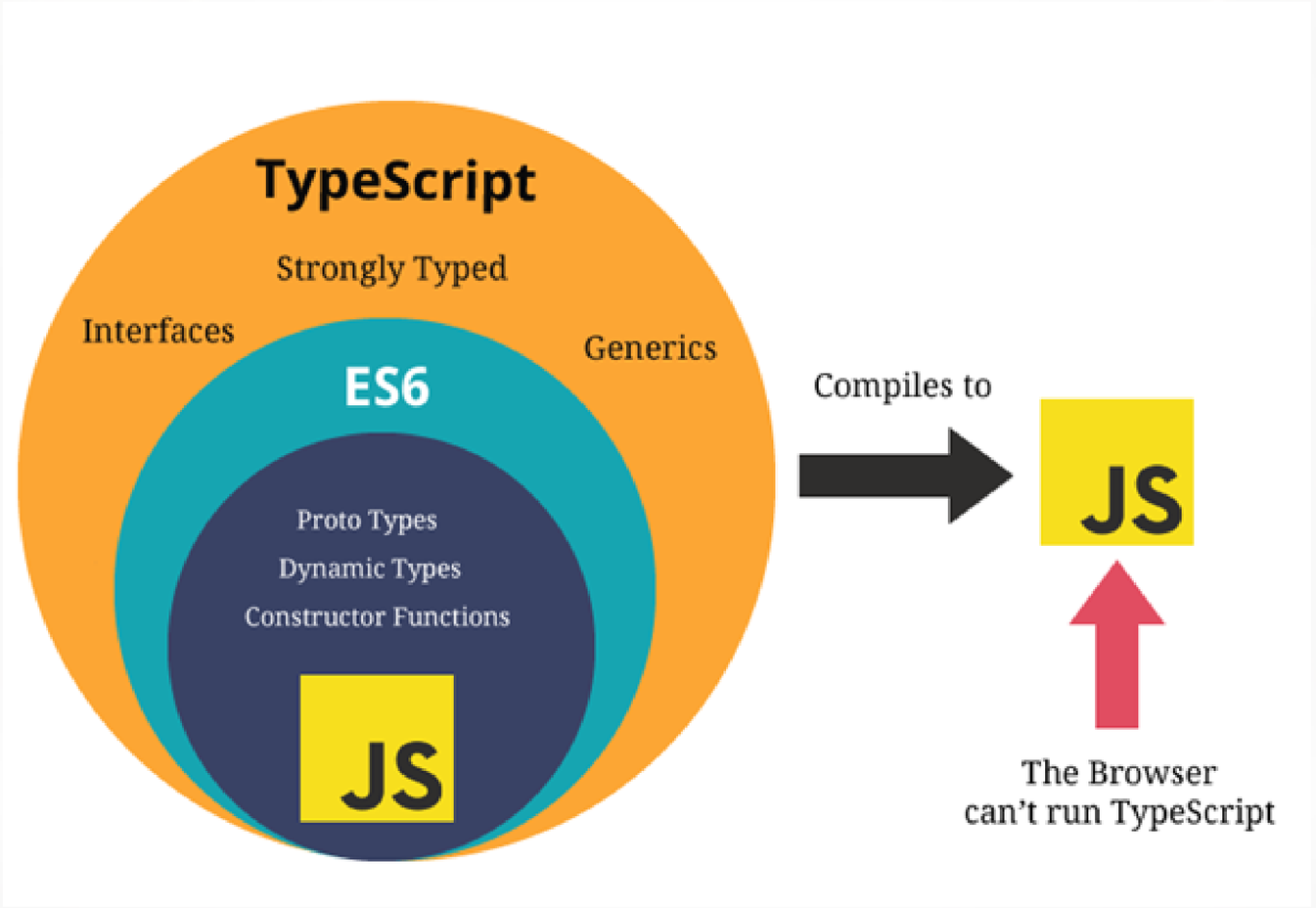
For typing arrays the syntax is the following:

```
let numbersArray: number[] = [1, 2, 3]; // We only accept numbers in this array
let numbersAndStringsArray: (number | string)[] = [1, "two", 3]; // Here we accept numbers and strings.
```

Tuples are arrays with fixed size and types for each position. They can be built like this:

```
let skill: [string, number];
skill = ["Programming", 5];
```

TypeScript's Compiler



- Every time we run our TypeScript file, TypeScript compiles our code and at that point, it checks the types. Only if everything is ok, the program runs. That's why we can get errors detected before program execution.

- TypeScript **transpiles** code into JavaScript.

- Browsers don't understand TypeScript. They understand JavaScript code. Hence, the TypeScript code needs to get compiled into JavaScript, and for that, you need the TypeScript compiler.

- We can also select to what "vesion" of JavaScript we want to transpile to, for example es4, es5, and so on.

How to Setup a TypeScript Project

- We'll need Node and NPM installed in our system.
- Once we're in the directory of our project, we first run **npm i typescript --save-dev**.
- Then we run **npx tsc --init**. This will initialize your project by creating a **tsconfig.json** file in your directory

We can then create a file with the **index.ts** extension and start writing our TypeScript code. Whenever we need to transpile our code to vanilla JS, we can do it by running **tsc <name of the file>.**

```
const personDescription = (name: string, city: string, age: number) =>
  `${name} lives in ${city}. he's ${age}. In 10 years he'll be ${age + 10}.`;
```

After running **tsc index.ts**, a new **index.js** file is automatically created in the same directory with the following content:

```
var personDescription = function (name, city, age) { return name + " lives in " +
```

```
city + ". he's " + age + ". In 10 years he'll be " + (age + 10) + "."; };
```

Is TypeScript worth it ?

- TypeScript can catch 15% of common bugs, improving code reliability.
- Enhanced code readability in teams makes it easier to understand and collaborate.
- Proficiency in TypeScript expands job opportunities due to its popularity.
- Learning TypeScript provides a deeper understanding of JavaScript and broadens your skill set.

Pros

- Static type-checking
- Improved scalability
- ES6/ES7 support
- Improved IntelliSense support
- Better error handling

Cons

- Stiff learning curve
- Increased complexity
- Slower compilation speeds
- Limited browser support

Conclusion

And that wraps it up! We've covered a lot about TypeScript today. From its basic syntax to its cool features like static typing and generics, it's clear that TypeScript brings some serious muscle to web development.

As we move forward, let's keep TypeScript in our toolkit. It's not just about writing code, it's about making our lives easier, our projects cleaner, and our teams happier.

Thanks for hanging in there with us. Let's keep exploring TypeScript together and see where it takes us next!

THANK YOU