TypeScript

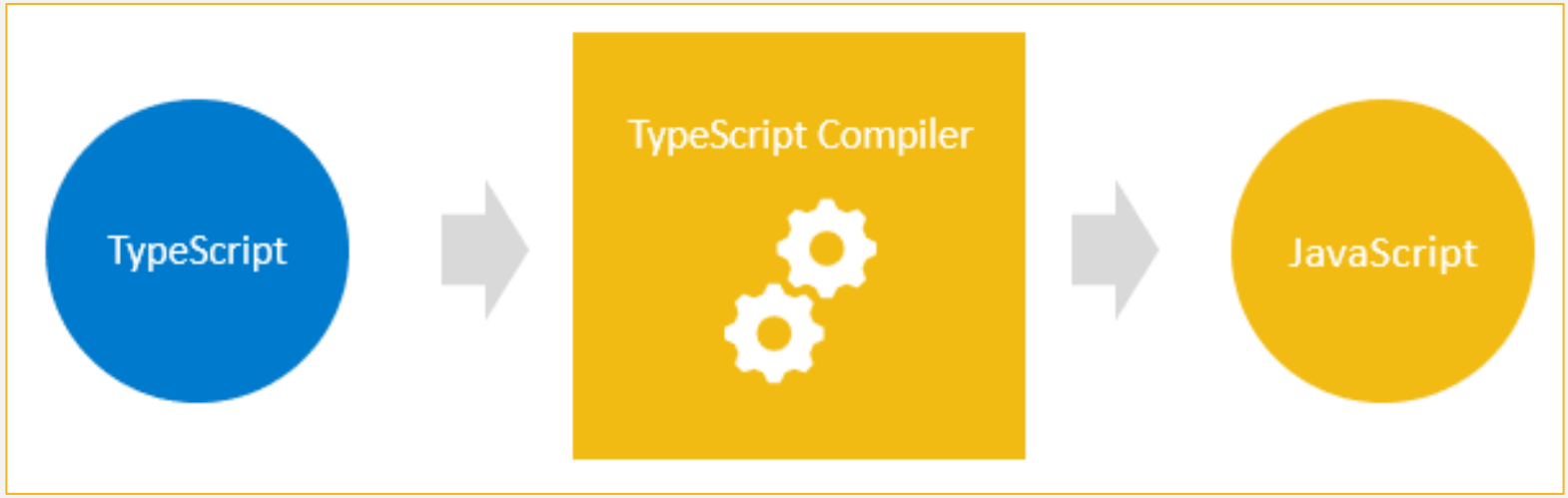
Getting Started

Content

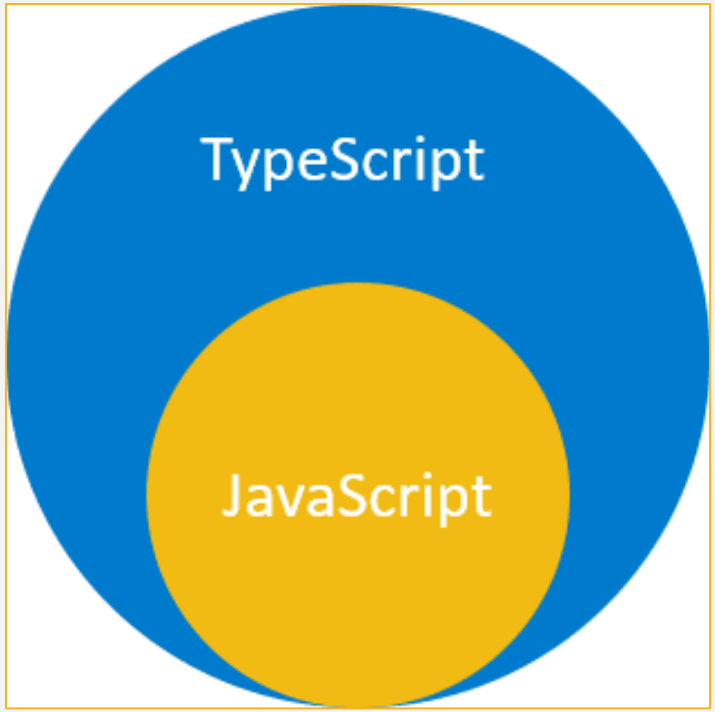
1. What is TypeScript?
2. TypeScript Hello World
3. Why TypeScript?

1. What is TypeScript?

- TypeScript is a super set of JavaScript.
- TypeScript builds on top of JavaScript. First, you write the TypeScript code. Then, you compile the TypeScript code into plain JavaScript code using a TypeScript compiler.
- Once you have the plain JavaScript code, you can deploy it to any environments that JavaScript runs.
- TypeScript files use the .ts extension rather than the .js extension of JavaScript files.



- TypeScript uses the JavaScript syntaxes and adds additional syntaxes for supporting Types.
- If you have a JavaScript program that doesn't have any syntax errors, it is also a TypeScript program. It means that all JavaScript programs are TypeScript programs. This is very helpful if you're migrating an existing JavaScript codebase to TypeScript.

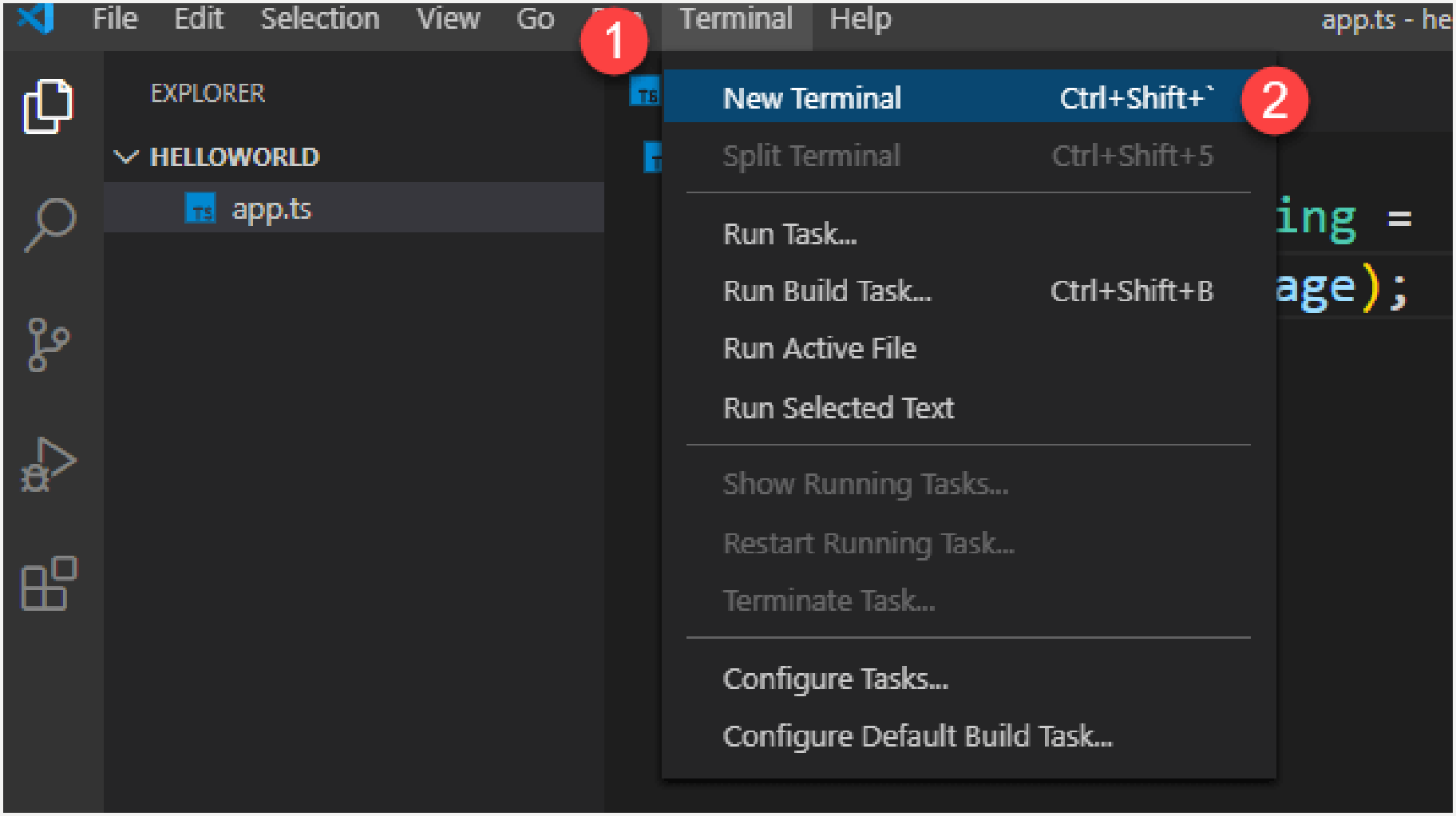


2. TypeScript Hello World

- First, create a new folder to store the code.
- Second, launch VS Code and open that folder.
- Third, create a new TypeScript file called app.ts
- Fourth, type the following source code in the app.ts file

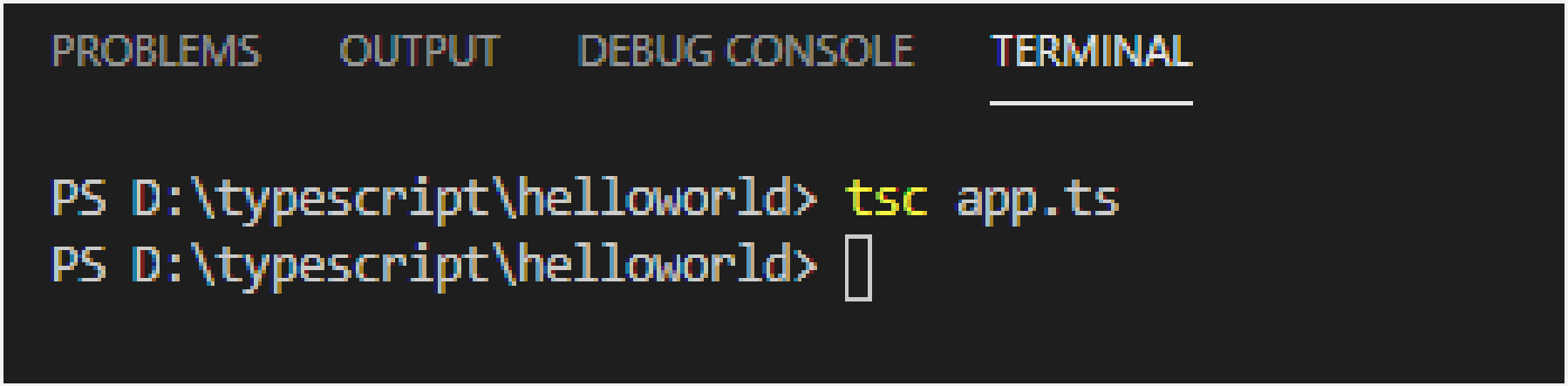
```
let message: string = 'Hello, World!';

console.log(message);
```

- Fifth, launch a new Terminal within the VSC by using the Ctrl+` or follow the menu Terminal > New Terminal

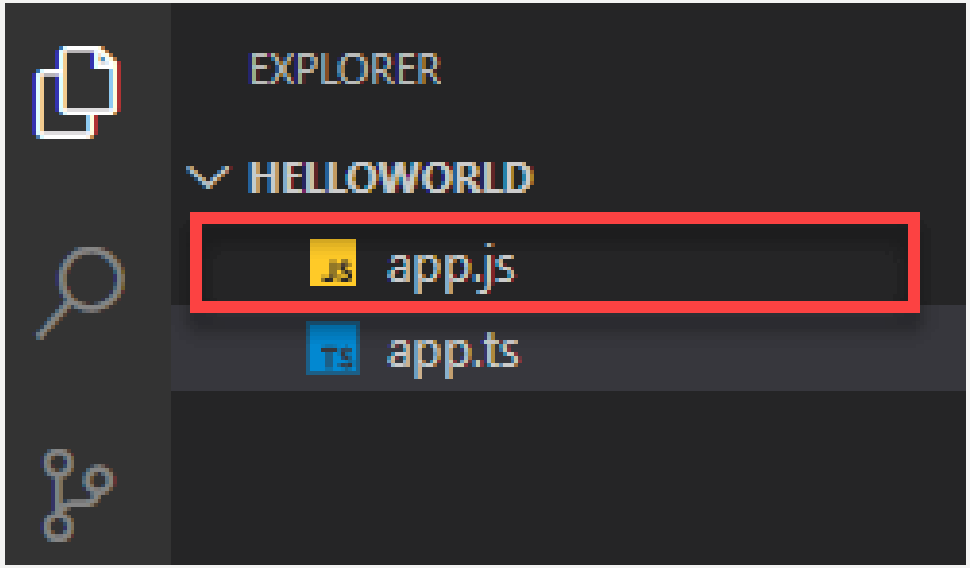


- Sixth, type the following command on the Terminal to compile the app.ts file

```
tsc app.ts
```



- If everything is fine, you'll see a new file called app.js is generated by the TypeScript compiler



- To run the app.js file in node.js, you use the following command

```
node app.js
```

TypeScript Hello World program in Web Browsers

- First, create a new file called index.html and include the app.js as follows

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>TypeScript: Hello, World!</title>
</head>
<body>
    <script src="app.js"></script>
</body>
</html>
```

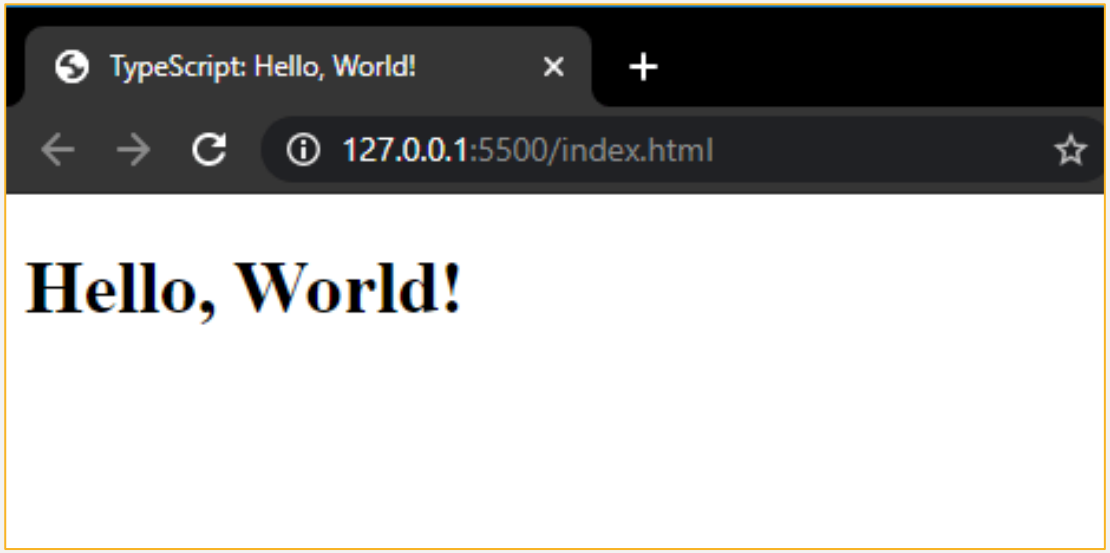
- Second, change the app.ts code to the following

```
let message: string = 'Hello, World!';

// create a new heading 1 element
let heading = document.createElement('h1');
heading.textContent = message;

// add the heading the document
document.body.appendChild(heading);
```

- Third, compile the app.ts file

```
tsc app.ts
```

- Fourth, open the Live Server from the VS code by right-mouse click the index.html and select the Open with Live Server option
- The Live Server will open the index.html with the following message



3. Why TypeScript?

- There are two main reasons to use TypeScript:
  - TypeScript adds a type system to help you avoid many problems with dynamic types in JavaScript.
  - TypeScript implements the future features of JavaScript a.k.a ES Next so that you can use them today.

3.1. Understanding dynamic type in JavaScript

3.2. Problems with dynamic types

3.3. How Typescript solves the problems of dynamic types

3.1. Understanding dynamic type in JavaScript

```
let box;
box = "hello";
box = 100;
```

```
let box;
console.log(typeof(box)); // undefined

box = "Hello";
console.log(typeof(box)); // string

box = 100;
console.log(typeof(box)); // number
```

3.2. Problems with dynamic types

- Suppose you have a function that returns a product object based on an id

```
function getProduct(id){
  return {
    id: id,
    name: `Awesome Gadget ${id}`,
    price: 99.5
  }
}
```

- The following uses the getProduct() function to retrieve the product with id 1 and shows its data

```
const product = getProduct(1);
console.log(`The product ${product.Name} costs $${product.price}`);
```

- Output

```
The product undefined costs $99.5
```

- The issue with this code is that the product object doesn't have the Name property. It has the name property with the first letter n in lowercase.

3.3. How Typescript solves the problems of dynamic types

- To fix the problem of referencing a property that doesn't exist on an object, you do the following steps:
- First, define the "shape" of the product object using an interface.

```
interface Product{
    id: number,
    name: string,
    price: number
};
```

- Second, explicitly use the Product type as the return type of the getProduct() function

```
function getProduct(id) : Product{
  return {
    id: id,
    name: `Awesome Gadget ${id}`,
    price: 99.5
  }
}
```

- When you reference a property that doesn't exist, the code editor will inform you immediately

```
const product = getProduct(1);
console.log(`The product ${product.Name} costs $${product.price}`);
```

- The code editor highlighted the following error on the Name property

```
const product = getProduct(1);
console.log(`The product ${product.Name} costs $${product.price}`);
```

THE END