Angular 8 : Online Class

18-August-2019

By: Sahosoft Solutions

Presented by : Chandan Kumar

Angular 8 : Online Class

TypeScript

By: Sahosoft Solutions

Presented by : Chandan Kumar

Typescript - if else

An if statement can include one or more expressions which return boolean.
If the boolean expression evaluates to true, a set of statements is then executed.

Example: if

```
if (true)
{
    console.log('This will always executed.');
}

if (false) {
    console.log('This will never executed.');
}
```

Typescript - if else Condition

An if else condition includes two blocks - if block and an else block.
If the if condition evaluates to true, then the if block is executed. Otherwise, the else block is executed.

Example: if

```
if (true)
{
    console.log("");
}
else{
     console.log("");
}
```

Remember: else cannot include any condition and it must follow if or else if conditions.

Typescript - else if Condition

The else if statement can be used after the if statement.

Example: if

```
if (true)
{
   console.log(“”);
}
Else if (false)
{
    console.log(“”);
}
Else {
    console.log(“”);
}
```

TypeScript - switch

The switch statement is used to check for multiple values and executes sets of statements for each of those values. A switch statement has one block of code corresponding to each value and can have any number of such blocks. When the match to a value is found, the corresponding block of code is executed.

Syntax:

```
switch(expression) {
   case constant-expression1: {
      //statements;
      break;
   }
   case constant_expression2: {
      //statements;
      break;
   }
   default: {
      //statements;
      break;
   }
}
```

TypeScript - switch

The following rules are applied on the switch statement:

1. The switch statement can include constant or variable expression which can return a value of any data type.
2. There can be any number of case statements within a switch. The case can include a constant or an expression.
3. We must use break keyword at the end of each case block to stop the execution of the case block.
4. The return type of the switch expression and case expression must match.
5. The default block is optional.

TypeScript - for Loops

There are three types of for loops:

1. for loop
2. for..of loop
3. for..in loop

for Loop

The for loop is used to execute a block of code a given number of times, which is specified by a condition.

Syntax:

```
for (first expression; second expression; third expression ) {
    // statements to be executed repeatedly
}
```

Here, the first expression is executed before the loop starts. The second expression is the condition for the loop to execute. And the third expression is executed after the execution of every code block.

for...of Loop

TypeScript includes the **for...of** loop to iterate and access elements of an array, list, or tuple collection. The for...of loop returns elements from a collection e.g. array, list or tuple, and so, there is no need to use the traditional for loop

for...in Loop

Another form of the for loop is for...in. This can be used with an array, list, or tuple. The for...in loop iterates through a list or collection and returns an index on each iteration.

Typescript - while Loop

The while loop is another type of loop that checks for a specified condition before beginning to execute the block of statements. The loop runs until the condition value is met.

Syntax:

```
while (condition expression) {
    // code block to be executed
}
```

The condition expression checks for a specified condition before running the block of code.

Typescript - do..while loop

The do..while loop is similar to the while loop, except that the condition is given at the end of the loop. The do..while loop runs the block of code at least once before checking for the specified condition. For the rest of the iterations, it runs the block of code only if the specified condition is met.

Syntax:

```
do {
// code block to be executed
}
while (condition expression);
```

Typescript - Data Modifiers

In object-oriented programming, the concept of 'Encapsulation' is used to make class members 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:

1. public
2. private
3. protected

public

By default, all members of a class in TypeScript are public. All the public members can be accessed anywhere without any restrictions.

private

The private access modifier ensures that class members are visible only to that class and are not accessible outside the containing class.

protected

The protected access modifier is similar to the private access modifier, except that protected members can be accessed using their deriving classes.

Typescript - Function

Functions are the primary blocks of any program. In JavaScript, functions are the most important part since the JavaScript is a functional programming language. With functions, you can implement/mimic the concepts of object-oriented programming like classes, objects, polymorphism, and, abstraction.

Functions ensure that the program is maintainable and reusable, and organized into readable blocks. While TypeScript provides the concept of classes and modules, functions still are an integral part of the language.

In TypeScript, functions can be of two types:

1. named
2. Anonymous

Named Functions

A named function is one where you declare and call a function by its given name.

Anonymous Function

An anonymous function is one which is defined as an expression. This expression is stored in a variable. So, the function itself does not have a name. These functions are invoked using the variable name that the function is stored in.

Typescript - Function

Function Parameters

Parameters are values or arguments passed to a function. In TypeScript, the compiler expects a function to receive the exact number and type of arguments as defined in the function signature.

If the function expects three parameters, the compiler checks that the user has passed values for all three parameters i.e. it checks for exact matches.

Optional Parameters

TypeScript has an optional parameter functionality. The parameters that may or may not receive a value can be appended with a '?' to mark them as optional.

Default Parameters

TypeScript provides the option to add default values to parameters. So, if the user does not provide a value to an argument, TypeScript will initialize the parameter with the default value.

Default parameters have the same behaviour as optional parameters. If a value is not passed for the default parameter in a function call, the default parameter must follow the required parameters in the function signature. Hence, default parameters can be omitted while calling a function.

However, if a function signature has a default parameter before a required parameter, the function can still be called, provided the default parameter is passed a value of undefined.

Typescript - Function

Arrow Function

Fat arrow notations are used for anonymous functions i.e for function expressions. They are also called lambda functions in other languages.

Syntax:

(param1, param2, ..., paramN) => expression

Using fat arrow (=>) we drop the need to use the 'function' keyword. Parameters are passed in the angular brackets <>, and the function expression is enclosed within the curly brackets {}.

Typescript - Rest Parameters

When the number of parameters that a function will receive is not known or can vary, we can use rest parameters.

In JavaScript, this is achieved with the "arguments" variable. However, with TypeScript, we can use the rest parameter denoted by ellipsis ....

We can pass zero or more arguments to the rest parameter. The compiler will create an array of arguments with the rest parameter name provided by us.

Remember, rest parameters must come last in the function definition, otherwise the TypeScript compiler will show an error. The following is not valid.