upGrad

Introduction to TypeScript

upGrad

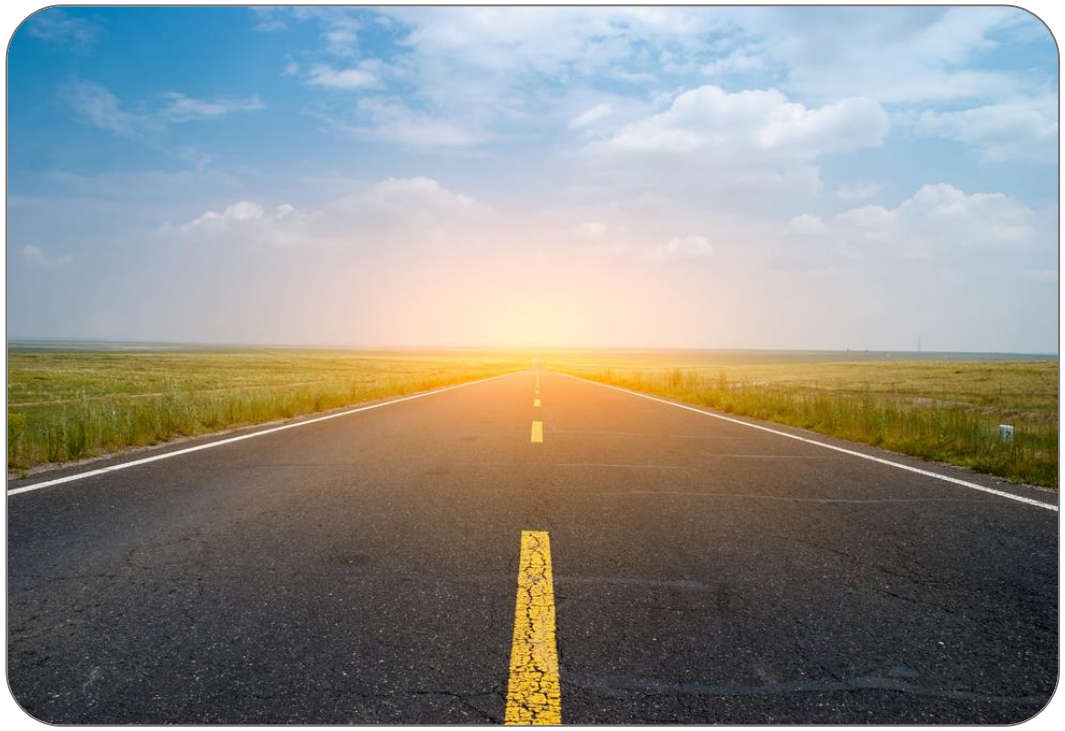
Course: Introduction to TypeScript

Lecture On: Functions in TypeScript

Instructor: Aquib Ajani

COURSE ROADMAP

- Introduction to TypeScript (TS)
- Operators, conditions and loops in TS
- **Functions in TS**
- Hands-on coding session - I
- Classes and interfaces in TS
- Type manipulation in TS
- Modules in TS
- Hands-on coding session - II



TODAY'S AGENDA

- TypeScript function
- Arrow/Lambda functions
- This parameters
- Function overloading

Functions in TS

FUNCTIONS IN TS

TypeScript Functions

Function are a set of instructions combined in a logical block to be executed when the function is called and typescript functions are no different. Writing functions makes code more readable and easy to maintain.

Several functions can be written, such as:

- Named function:

  While calling a named function, functions can be called by their names.

  Example:

```
function hello(){
     console.log(“hello”)
};
```

FUNCTIONS IN TS

TypeScript Functions

- Anonymous function: Functions that do not have a name or do not bind to any name are called anonymous functions.

  Example:

```
var a = function(){
     console.log(“hello”)
};
```

- The anonymous function shown above is called as follows:

```
a();
```

- Since variable a is taking the complete function as input, the function will be called and the output will be ‘**hello**’.

FUNCTIONS IN TS

TypeScript Functions

Function types:

- When a function is called, the value it returns defines the type of function.

Type of a function can be identified in following ways:

- Typing the function (explicitly writing the function type)
- Inferring the function type

FUNCTIONS IN TS

TypeScript Functions

Typing the function:

Example:

```
function hello(): string{ //defined return type of function
      return “hello”;
}
```

- **Inferring the types**: If you do not explicitly define the return type of a function, it gets automatically inferred.

  Example:

```
function hello(){ //defined return type of function
        return “hello”;  // return type is string and thus
it will inferred as return type of function to be string
}
```

FUNCTIONS IN TS

TypeScript Functions

Function parameters:

- When you need to pass some values to a function, you use parameters to be passed and are called function parameters.

  Example:

```
function hello(name:string): string{ //defined type
of parameter name
      return “hello”+ name;
}
```

FUNCTIONS IN TS

TypeScript Functions

Optional parameters:

- You can mark any parameter as optional if it is not mandatory to the function and that can be done by a '?' symbol.

  Example:

```
function  hello(name?:string){ //? Makes the
parameter optional
      return "hello"+ name;
}
```

FUNCTIONS IN TS

TypeScript Functions

Default parameters:

- When you pass some default values to the function parameter, it becomes optional, and if not passed during function call, it takes default values.

  Example:

```
function  function hello(name:string = “Parth”):
string{ //given default value to the parameter.
       return “hello”+ name;    // helloParth
}
```

FUNCTIONS IN TS

TypeScript Functions

Rest parameters:

- If you are uncertain about the number of parameters to be passed, we can use a rest parameter (…) to accept as many parameters coming but the only restriction is that they all should be of same type.

Example:

```
function  function hello(…name:string[]){ //? Makes
the parameter optional
        for(I=0;i<name.length;i++)
              console.log(name[i]);
}
```

FUNCTIONS IN TS

Arrow/Lambda Functions

Arrow/Lambda functions:

- An arrow function is basically an anonymous function that is also a shorthand of the normal function, but the major difference is that it does not have ‘this’ of its own and the function keyword is removed.
- It is represented by a fat arrow (=>).
- It has three main parts:
  - Parameters
  - Fat arrow (=>)
  - Statements
- An important use of arrow functions is to write concise functions that do not require ‘this’ for any reference.

FUNCTIONS IN TS

Arrow/Lambda Functions

Arrow/Lambda expressions:

- When the complete function is expressed in a single line of code, it is called as a lambda expression.

  Example:

```
var foo = (x:number)=>10 + x
console.log(foo(100))      //outputs 110
```

FUNCTIONS IN TS

Arrow/Lambda Functions

Example:

```
var sum  = (num:number) => {
    console.log(10+num);
}
```

- In the above example, the portion highlighted in gold is the parameter that is being passed, the portion in orange is the statement or the code block.
- The portion highlighted in green is a variable to which this function is assigned (as the arrow function itself is an anonymous function).

FUNCTIONS IN TS

Arrow/Lambda Functions

Pros and cons of Arrow/Lambda

Pros:

- It reduces a lot of code and makes it more readable.
- The greatest advantage of having contextual ‘this’, no longer need to ‘bind’ the functions any more.
- Most modern browsers support Arrow functions out of the box. Although, it is still advisable to use a transpiler like Bable to polyfill for backward compatibility.

Cons:

- It is not to be used in object methods as it has no ‘this’ and will refer to parent for the same.
- It is not to be used in a callback function with dynamic context based on values.

FUNCTIONS IN TS

This Parameter

This parameter:

- 'This' parameter is basically a context of the object of the function when invoked. It is one of the important concepts of JavaScript itself. This parameter is an implicit parameter and is passed automatically when the function is invoked.

Example:

```
const test = {
  prop: 42,
  func: function() {
    return this.prop;
  },
};

console.log(test.func());
// expected output: 42
```

FUNCTIONS IN TS

This Parameter

This parameter in callbacks:

- Let's assume you have a function with a callback function.

Example:

```
function outerFunc(callback){
      callback();
}
function getThis(){
       console.log(this);
}
```

Now if you call outerFunc to pass the getThis function as the parameter, it will done as follows:

```
outerFunc(getThis);
```

This function will return the Window object as the outerFunc function is written in global space and thus 'this' is referred to the window object.

To conclude: Callback functions are at the mercy of whichever higher order function they are passed into.

FUNCTIONS IN TS

Function Overloading

Function overloading:

- When you have more than one function with the same name but different parameters, while calling the function, the function is chosen based on the parameters passed and this concept is known as function overloading.

Example:

```
function sum(a,b,c){
     return a+b+c;
}
function sum(a,b){
     return a+b;
}
```

FUNCTIONS IN TS

Function Overloading

Function overloading:

- In TS, you have the same concept but with different implementation. When you have more than one function with same name and same parameters but different data types of the parameters, while calling the function, the function is chosen based on the parameters passed. This concept is known as function overloading in TS.

Example:

```
function add(a:string, b:string):string;
function add(a:number, b:number): number;
function add(a: any, b:any): any {
    return a + b;
}
```

above function is called as follow:

```
add("Hello","Parth"); // returns "Hello Parth"
Add (3,4); // return 7
```

Note - Function overloading with different number of parameters and types with same name is not supported.

FUNCTIONS IN TS

Function Overloading

Advantages of function overloading:

- Compiling a code is too fast since it saves memory space.
- Maintaining a code is easy.
- It facilitates code reusability, which saves time and effort.
- Readability of the program increases.

FUNCTIONS IN TS

Optional chaining

- Optional chaining is denoted by '?' and helps to check nested chain of attributes in an object.
- You can descend in an object with simultaneous check of null or undefined with the help of '?'.

Example:

```
function serializeJSON(value: any, options?: SerializationOptions) {
 const indent = options?.formatting?.indent; // optional chaining
 return JSON.stringify(value, null, indent);
}
```

Two more types are as follows:

- ?.() => represents calling the function if it exists.
- ?.[] => represents finding the value of the key, if the key exists.

CODING QUESTION

- Write a function to filter students who have obtained more than 80% in class.
- Link to the GitHub solution

KEY TAKEAWAYS

- TypeScript function
- Arrow/Lambda functions
- This parameters
- Function overloading

TASKS TO COMPLETE AFTER THE SESSION

MCQs
Coding Questions

upGrad

Thank you!