upGrad

Introduction to TypeScript

upGrad

Course: Introduction to TypeScript

Lecture On: Operators, Conditions and Loops in TS

Instructor: Aquib Ajani

COURSE ROAD MAP

- 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 TypeScript
- Modules in TS
- Hands-on Coding Session - II

TODAY'S AGENDA

- TypeScript Operators
- Conditional Statements in TypeScript
- Looping in TypeScript
- Problem-Solving

Operators, Conditions and Loops in TS

OPERATORS, CONDITIONS AND LOOPS IN TS

TypeScript Operators

Arithmetic Operators:
There are various arithmetic operators being used for the operations (assuming var a and b have values 10 and 5, respectively). They are given below

Operator	Description	Example
+ (Addition)	Returns the sum of the operands	a + b is 15
- (Subtraction)	Returns the difference of the values	a – b is 5
* (Multiplication)	Returns the product of the values	a * b is 50
/ (Division)	Performs a division operation and returns the quotient	a / b is 2
% (Module)	Performs a division and returns the remainder	a % b is 0
++ (increment)	Increment the value of the variable by one	a ++ is 11
-- (Decrement)	Decrement the value of the variable by one	a -- is 9

OPERATORS, CONDITIONS AND LOOPS IN TS

TypeScript Operators

Arithmetic Operators:
Examples:

```
var a : number = 12;
var b : number = 10;
console.log(a+b);
console.log(a-b);
console.log(a*b);
console.log(a/b);
console.log(a%b);
console.log(a-b);
console.log(a++);
console.log(b—);
```

OPERATORS, CONDITIONS AND LOOPS IN TS

TypeScript Operators

Comparison Operators:
There are various comparison operators being used for the comparison of two variables. Types of comparison operators are given below (A = 10 and B = 20)

Operator	Description	Example
>	Greater than	(A>B) is False
<	Lesser than	(A<B) is True
>=	Greater than or equal to	(A>=B) is False
<=	Leaser than or equal to	(A<=B) is True
==	Equality	(A==B) is false
!=	Not equal	(A!=B) is true

OPERATORS, CONDITIONS AND LOOPS IN TS

TypeScript Operators

Comparison Operators:
Examples:

```
var a : number = 12;
var b : number = 10;

console.log(a>b);
console.log(a<b);
console.log(a>=b);
console.log(a<=b);
console.log(a==b);
console.log(a!=b);
```

OPERATORS, CONDITIONS AND LOOPS IN TS

TypeScript Operators

Logical Operators:
There are various logical operators being used for the comparison of two variables and to combine two different conditions and return boolean value. Types of logical operators are given below (A = 10 and B = 20)

Operator	Description	Example
&& (And)	The operator returns true only if all the expression specified return true	(A> 10 && B > 10) is False
|| (OR)	The operator returns true if at least one of the expressions specified return true	(A> 10 || B > 10) is True
! (NOT)	The operator returns the inverse of the expression's result. For E.g.: !(>5) returns false	!(A>10) is True

OPERATORS, CONDITIONS AND LOOPS IN TS

TypeScript Operators

Logical Operators:
Examples:

```
var a : number = 12;
var b : number = 10;

console.log(a>10 && b>12);
console.log(a<10 || b>12);
console.log!(a>10);
```

OPERATORS, CONDITIONS AND LOOPS IN TS

TypeScript Operators

Bitwise Operators:

These perform operations on every bit of arguments.
Here are the types of bitwise operators
(A=2 and B=3)

Operator	Description	Example
& (Bitwise AND)	It performs a Boolean AND operation on each bit of its integer arguments.	(A & B) is 2
| (BitWise OR)	It performs a Boolean OR operation on each bit of its integer arguments.	(A | B) is 3
^ (Bitwise XOR)	It preforms a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.	(A^B) is 1
~ (Bitwise Not)	It is a unary operator and operates by reversing all the bits in the operand.	(-B) is -4
<< (left Shit)	It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one positions is equivalent to multiplying by 4, and so on.	(A<<1) is 4
>> (Right Shift)	Binary Right Shift Operator. The left operator's value is moved right by the number of bits specified by the right operand.	(A>>1) is 1
>>> (Right shift with Zero)	This operator is just like the >> operator, expect that the bits shifted in on the left are always zero.	(A>>>1) is 1

OPERATORS, CONDITIONS AND LOOPS IN TS

TypeScript Operators

Assignment Operators:
These operators assign updated values to the variables. Types of assignment operators are given below

Operator	Description	Example
= (Single Assignment)	Assign values from the right side operand to the left side operand.	C= A + B will assign the value of A + B into C
+=(Add and assignment)	It adds the right operand to the left operand and assigns the result to the left operand.	C += A is equivalent to C = C- A
*= (Subtract and Assignment)	It subtracts the right operand from the left operand and assigns the results to the left operand.	C -= A is equivalent to C = C- A
*=(Multiply and Assignment)	It multiplies the right operand with left operand and assigns the results to the left operand.	C *=A is equivalent to C = C*A
/= (Divide and Assignment)	IT divides the left operand with the right operand and assigns the result to the left operand.	

OPERATORS, CONDITIONS AND LOOPS IN TS

TypeScript Operators

Assignment Operators:
Examples:

```
var a : number = 12;
var b : number = 10;
var c : number = a + b;
console.log(c);
c + = a;
console.log(c);
c -=a;
console.log(c);
c *=a;
console.log(c);
```

OPERATORS, CONDITIONS AND LOOPS IN TS

TypeScript Operators

Conditional Operator (?):
Sample:

check ? expression1 : expression2

Also known as ternary operator. It checks for condition, and based on the boolean value returned, it proceeds further with expression1 if true and expression2 if false

Example:
var a : number = 2;
a < 3 ? “yes” : “no”

OPERATORS, CONDITIONS AND LOOPS IN TS

TypeScript Operators

Concatenation Operator (+):

When this operator is applied to strings, it appends the second string to the first

Example:

```
var a : string = “hello”+”world”;
console.log(a);
```

OPERATORS, CONDITIONS AND LOOPS IN TS

TypeScript Operators

Type Operator (+):

Typeof is a unary operator, and it returns the data type of the operand

Example:

```
var a = 3;
console.log(typeof a);
```

TYPESCRIPT CONDITIONS

Conditional Statements in TypeScript

- Conditional statements help TypeScript perform different actions based on certain given conditions
- There are two types of conditional statements in TS: **if...else** and **switch()**

if ()

else ()

switch ()

case 1:

case 2:

TYPESCRIPT CONDITIONS

If…Else

- The first variant is the **if** conditional statement. It checks whether a specified condition in a parameter is true or false. If it is true, then it executes the code corresponding to the condition, and if it is false, it does not execute the condition inside the block
- Another variant is the **if…else** conditional statement. If the condition in the if statement is true, then the code will execute that condition. However, if it is false, then it will execute the else block

```
var x = 2;

if(x % 2 == 0) {
    console.log("The number is
even");
}
//The number is even
```

```
var x = 3;

if(x % 2 == 0) {
    console.log("The number is
even");
} else {
    console.log("The number is odd");
}
//The number is odd
```

```
var x = "Hello";

if(x % 2 == 0) {
    console.log("The number is
even");
} else if (x % 2 == 1){
    console.log("The number is odd");
}  else  {
    console.log("x is not a number");
}
//x is not a number
```

TYPESCRIPT CONDITIONS

Switch

- The **switch** statement can be used to perform different actions based on different conditions. In the switch statement's parentheses, an expression needs to be added. The answer to that expression evaluation will be one of the **cases** inside the switch statement. If none of the cases match, then the code should go to **default**. The switch statement is like a waterfall: if the first case matches the value, then it executes the code written inside the first case, but it will go to all the cases and execute statements in them. To avoid this, every case should end with a **break**. The **break** keyword stops the execution at that point and comes out of the switch

```
var x = 3;
switch(x - 2) {
    case 0:
        console.log("The answer is zero");
        break;
    case 1:
        console.log("The answer is one");
        break;
    case 2:
        console.log("The answer is two");
        break;
    default:
        console.log("x is not a number");
}
// The answer is one
```

Poll 8 (15 Sec)

What will be the output of the following program in the console?

1. 'Play Beethoven'
2. 'Play Bob Dylan'
3. Invalid operation
4. No output

```
var expr = 'electric guitar';
switch (expr){
   case 'piano':
      console.log('Play Beethoven');
      break;
   case 'acoustic guitar':
      console.log('Play Bob Dylan');
      break;
}
```

Poll 8 (Answer)

What will be the output of the following program in the console?

1. 'Play Beethoven'
2. 'Play Bob Dylan'
3. Invalid operation
4. **No output**

```
var expr = 'electric guitar';
switch (expr){
   case 'piano':
      console.log('Play Beethoven');
      break;
   case 'acoustic guitar':
      console.log('Play Bob Dylan');
      break;
}
```

TypeScript Looping

TYPESCRIPT LOOPING

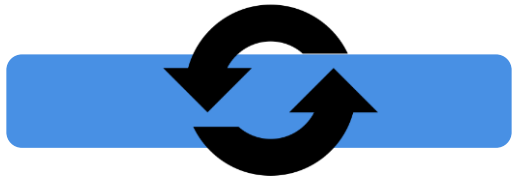
Introduction to Loops

- Loops can execute a block of code several times
- In TypeScript, there are three different methods for looping: **for(), while()** and **do...while()**

for *(condition)*
{

}

while *(condition)*
{



}

do
{

} while *(condition)*

TYPESCRIPT LOOPING

The for Loop

- Let's assume that you have an array and want to print every element in the array on a different line. You would have to run the same printing statement again and again, with different indices of the array

```
var students = [ "Rajesh", "Sheldon", "Leonard", "Howard"];

var text += students[0] + "\n";
text += students[1] + "\n";
text += students[2] + "\n";
text += students[3] + "\n";
console.log(text);
```

```
Rajesh
Sheldon
Leonard
Howard
```

Output

- In this case, it was a small array of four elements. However, if you have an array of 100 elements, then the process becomes tedious. This is where the for loop comes into the picture
- The syntax of a for loop has three statements: **statement1**, which is used for initialising variables (it is executed before the execution of the code block); **statement2**, which is the for condition (it defines the condition for executing the code block) and **statement3**, which is for incrementing/decrementing the value initialised in statement1 (it is executed every time the code block is executed)

```
for(initialization; condition; updation) {
   //code block to execute
}
```

TYPESCRIPT CONDITIONS

The for Loop

```
var students = [ “Rajesh”, “Sheldon”, “Leonard”, “Howard”];

for(var i = 0; i < students.length; i++) {

        var text += students[i] + "\n";

}
```

- As shown in the given code, the first step is to initialise the variable i with the value 0
- Next, it will check the condition in statement 2. If the value is true, then it will execute the code block. Since 0 < length of student array (=4), the condition is true
- Then, the statement in the code block will be executed, where students[i], which will be students[0] as i = 0, will be added to text. Hence, “Rajesh” is appended to text
- Then, an increment will happen as part of the updation statement inside the for loop; so, the value of i will become 1
- Next, the condition will be checked again, and if it is true, then it will execute the code block. If it is false, which means that the entire array has been traversed, then it will stop the execution and come out of the for loop

TYPESCRIPT CONDITIONS

The for…of and for…in Loop

- The **for …of** loop returns elements from a collection like an array or tuple, so there is no requirement of using the for loop in a traditional way

```
let arr = [10, 20, 30, 40];

for (var val of arr) {
  console.log(val); // prints values: 10, 20, 30, 40
}
```

- The **for …in** loop returns an index from a collection like an array or tuple, so you have to access elements from the array or list using the traditional method like the one below for an array

```
let arr = [10, 20, 30, 40];

for (var index in arr) {
  console.log(index); // prints indexes: 0, 1, 2, 3

  console.log(arr[index]); // prints elements: 10, 20, 30, 40
}
```

TYPESCRIPT LOOPING

The While Loop

- The **while** loop loops through a block of code as long as the specified condition holds true

```
while(condition) {
   //code block to execute
}
```

- In simple words, the code block will keep executing as long as the condition mentioned in the parentheses does not turn false. As shown in the example below, the code will keep adding values to text as long as the value of i remains below 10. An important point to note here is that if the value of i is never incremented, then the loop will keep executing an infinite number of times, which will crash your browser. So, it is important to ensure that there is always an increment/decrement of the value

```
var i = 0, text="";
while(i < 10) {
      text += "The number is " + i + "\n";
      i++;
}
console.log(text);
```

TYPESCRIPT LOOPING

The Do…While Loop

- The **do…while** loop is similar to the while loop. In the while loop, if the condition is not met, then the code block will not be executed at all. However, in the do…while loop, the code block will be executed once, and then the condition will be checked. If the condition is true, then the code block will be executed repeatedly until the condition becomes false

```
do {
   //code block to execute
}
while(condition);
```

- In the first example below, the condition is false. However, the code will be executed once, and outputting text will give "The number is 0". In the second example, as the condition is true, the code block will be executed until the condition becomes false. As with the while loop, if the variable is not incremented/decremented, then the browser will crash because of an infinite loop

```
var i = 0, text="";
do {
      text  +=  "The number is "  +  i;
      i++;
}
while(i  >  10);
console.log(text);
```

```
var i = 0, text="";
do {
      text  +=  "The number is "  +  i;
      i++;
}
while(i  <  10);
console.log(text);
```

PROBLEM-SOLVING IN TS

Problem-Solving

Write a TypeScript program to compute the greatest common divisor (GCD) of two positive integers.

- GitHub link to the solution

PROBLEM-SOLVING IN TS

Problem-Solving

Write a JavaScript conditional statement to find the sign of product of three numbers.

- GitHub link to the solution

PROBLEM-SOLVING IN TS

Problem-Solving

Write a JavaScript code to convert the given input number into words.

- GitHub link to the solution

PROBLEM-SOLVING IN TS

Problem-Solving

Write a JavaScript code that takes a number as an input and prints the following pattern accordingly.

Enter the size: 7

```
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
```

GitHub link to the solution

KEY TAKEAWAYS

- TypeScript Operators
- Conditional Statements in TypeScript
- Looping in TypeScript
- Problem-Solving

TASKS TO COMPLETE AFTER THE SESSION

MCQs
Coding Questions

upGrad

Thank You!