3

Operators and Control Flow Statements

Objectives

After completing this lesson, you should be able to do the following:

- Describe uses of literals and Typescript operators
- Identify valid operator categories and operator precedence
- Use string object literals and the concatenation operator
- Use decision-making constructs
- Perform loop operations
- Write `switch` statements

Operators

- Operators manipulate data and objects.
- Operators take one or more arguments and produce a value.
- There are 44 different operators.
- Some operators change the value of the operand.

Categories of Operators

There are five types of operators:

- Assignment
- Arithmetic
- Integer bitwise
- Relational
- Boolean

Using the Assignment Operator

The result of an assignment operation is a value and can be used whenever an expression is permitted.

- The value on the right is assigned to the identifier on the left:

```
var1 : number = 0, var2 = 0;
var1 = 50;          // var1 now equals 50
var2 = var1 + 10; // var2 now equals 60
```

- The expression on the right is always evaluated before the assignment.
- Assignments can be strung together:

```
var1 = var2 = var3 = 50;
```

Working with Arithmetic Operators

- Used to perform basic arithmetic operations
- Work on numeric variables and literals

```
int a, b, c, d, e;
a = 2 + 2;     // addition
b = a * 3;     // multiplication
c = b - 2;     // subtraction
d = b / 2;     // division
e = b % 2;     // returns the remainder of division
```

Incrementing and Decrementing Values

- The ++ and -- operators increment and decrement by 1, respectively:

```
var1: number = 3;
var1++;            // var1 now equals 4
```

- The ++ and -- operators can be used in two ways:

```
var1 : number = 3, var2 : : number = 0;
var2 = ++var1;    // Prefix:  Increment var1 first,
                  //           then assign to var2.
var2 = var1++;    // Postfix: Assign to var2 first,
                  //           then increment var1.
```

Relational and Equality Operators

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to

```
var1 : number = 7, var2 : number = 13;
res : boolean = true;
res = (var1 == var2);       // res now equals false
res = (var2 > var1);        // res now equals true
```

Conditional Operator (? :)

- Useful alternative to `if…else`:

```
boolean_expr ? expr1 : expr2
```

- If *`boolean_expr`* is `true`, the result is *`expr1`*; otherwise, the result is *`expr2`*:

```
val1 : number = 120, val2 = 0;
highest : number ;
highest = (val1 > val2) ? val1 : val2;
console.log("Highest value is " + highest);
```

Logical Operators

Results of Boolean expressions can be combined by using logical operators:

`&&`	`&`	AND (with or without short-circuit evaluation)
`||`	`|`	OR (with or without short-circuit evaluation)
`^`		exclusive OR
`!`		NOT

```
var0 : number = 0, var1 : number = 1, var2 : number = 2;
res : boolean = true;
highest = (val1 > val2)? val1 : val2;
res = !res;
```

Compound Assignment Operators

An assignment operator can be combined with any conventional binary operator:

```
total : number =0, num : number = 1;
percentage : number = .50;
…
total  = total + num;    // total is now  1
total += num;            // total is now  2
total -= num;            // total is now  1
total *= percentage;     // total is now .5
total /= 2;             // total is now 0.25
num %= percentage;      // num is now 0
```

Operator Precedence

Order	Operators	Comments	Assoc.
1	`++ -- + - ~ ! (`*type*`)`	Unary operators	R
2	`* / %`	Multiply, divide, remainder	L
3	`+ - +`	Add, subtract, add string	L
4	`<< >> >>>`	Shift (>>> is zero-fill shift)	L
5	`< > <= >= instanceof`	Relational, type compare	L
6	`== !=`	Equality	L
7	`&`	Bit/logical AND	L
8	`^`	Bit/logical exclusive OR	L
9	`|`	Bit/logical inclusive OR	L
10	`&&`	Logical AND	L
11	`||`	Logical OR	L
12	`?:`	Conditional operator	R
13	`=` *op=*	Assignment operators	R

More About Operator Precedence

- Operator precedence determines the order in which operators are executed:

```
var1 : number = 0;
var1 = 2 + 3 * 4;     // var1 now equals 14
```

- Operators with the same precedence are executed from left to right (see

```
var1 : number = 0;
var1 = 12 - 6 + 3;    // var1 now equals 9
```

Use parentheses to override the default order.

Concatenating Strings

The + operator creates and concatenates strings:

```
name : string = "Jane ";
lastName : string = "Hathaway";
fullName : string;
name = name + lastName;     // name is now
                            //"Jane Hathaway"
                            //    OR
name += lastName;           // same result
fullName = name;
```

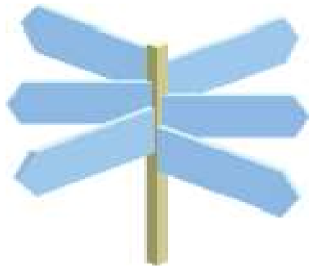
Basic Flow Control Types

Flow control can be categorized into four types:

Sequential

Iteration

Selection



Transfer

Using Flow Control in TypeScript

- Each simple statement terminates with a semicolon (;).
- Group statements by using braces { }.
- Each block executes as a single statement in the flow of control structure.

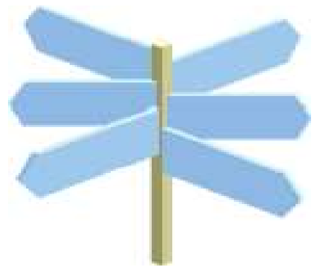
```
{
  finished: boolean = true;
  console.log("i = " + i);
  i++;

}
```

`if` Statement

General:

```
if ( boolean_expr )
    statement1;
[else
    statement2;]
```



Examples:

```
if (i % 2 == 0)
   console.log("Even");
else
   console.log("Odd");
…
```

```
if (i % 2 == 0) {
    console.log(i);
    console.log(" is even");
}
```

Nested `if` Statements

```
if (speed >= 25)
  if (speed > 65)
       console.log("Speed over 65");
  else
       console.log("Speed >= 25 but <= 65");
  else
       console.log("Speed under 25");
```

```
if (speed > 65)
       console.log("Speed over 65");
else if (speed >= 25)
       console.log("Speed greater… to 65");
     else
       console.log("Speed under 25");
```

Guided Practice: Spot the Mistakes

1

```
x : number = 3, y: number = 5;
if (x >= 0)
  if (y < x)
        console.log("y is less than x");
else
        console.log("x is negative");
```

2

```
x: number = 7;
if (x = 0)
 console.log("x is zero");
```

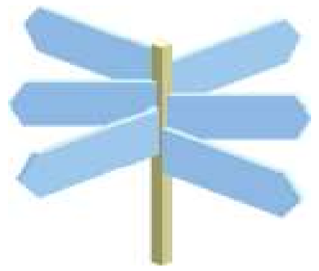
3

```
x: number = 14, y :number= 24;
if ( x % 2 == 0  &&  y % 2 == 0 );
        console.log("x and y are even");
```

switch Statement

```
switch ( integer_expr ) {

    case constant_expr1:
        statement1;
        break;
    case constant_expr2:
        statement2;
        break;
    [default:
        statement3;]
}
```



- The `switch` statement is useful when selecting an action from several alternative integer values.
- `Expr` must be `number char, or` **`String`**.

Strings in Switch

Without String Usage in Typescript

```
monthNameToDays(s: string,
 year: number): number {
   if(s.equals("April") ||
      s.equals("June") ||
      ... )
         return 30;
   if(s.equals("January") ||
      s.equals("March") ||
      ... )
         return 31;
   if(s.equals("February"))
      ...
   else
      ...
   }
 }
```

With String Usage in Typescript

```
monthNameToDays(s: string,
    year: number): number{
        switch(s) {
            case "April":
            case "June":
            ...
                return 30;
            case "January":
            case "March":
            ...
                return 31;
            case "February":
                ...
            default
                ...
        }
    }
```

More About the `switch` Statement

- `case` labels must be constants.
- Use `break` to jump out of a switch.
- You should always provide a default.

```
switch (choice) {
  case 37:
     console.log("Coffee?");
     break;

  case 45:
     console.log("Tea?");
     break;

  default:
     console.log("???");
     break;
}
```

Looping in Java

- There are three types of loops in Java:
  - `while`
  - `do…while`
  - `for`
- All (counter-controlled) loops have four parts:
  - Initialization
  - Body
  - Increment
  - Termination

while Loop

while is the simplest loop statement and contains the following general form:

```
while ( boolean_expr )
     statement;
```

Example:

```
i: number = 0;
while (i < 10) {
   console.log("i = " + i);
   i++;
}
```

`do...while` Loop

`do...while` loops place the test at the end:

```
do
     statement;
while ( termination );
```

Example:

```
i: number = 0;
do {
  console.log("i = " + i);
  i++;
} while (i < 10);
```

`for` Loop

`for` loops are the most common loops:

```
for ( initialization; termination; increment )
    statement;
```

Example:

```
for (i: number = 0; i < 10; i++)
    console.log(i);
```

More About the `for` Loop

- Variables can be declared in the initialization part of a `for` loop:

```
for (i: number = 0; i < 10; i++)
    console.log("i = " + i);
```

- Initialization and increment can consist of a list of comma-separated expressions:

```
for (i: number = 0, j: number = 10; i < j; i++, j--) {
        console.log("i = " + i);
        console.log("j = " + j);
}
```

Guided Practice: Spot the Mistakes

1

```
x: number = 10;
while (x > 0);
 console.log(x--);
console.log("We have lift off!");
```

2

```
x: number = 10;
while (x > 0)
 console.log("x is " + x);
  x--;
```

3

```
sum: number = 0;
for (; i < 10; sum += i++);
console.log("Sum is " + sum);
```

MentorLabs℠

`break` Statement

- Breaks out of a loop or `switch` statement
- Transfers control to the first statement after the loop body or `switch` statement
- Can simplify code but must be used sparingly

```
…
while (age <= 65) {
    balance = (balance+payment) * (1 + interest);
    if (balance >= 250000)
        break;
    age++;
}
…
```

`continue` Statement

- Skips the iteration of a loop
- Moves on to the next one

```
…
for (i: number =0; i<=10; i++) {
//skips the print statement if i is not even
       if(i % 2 != 0) {
              continue;
       }
//prints the integer "i" followed by a space
console.log(i + ' ');
}
 …
```

Summary

In this lesson, you should have learned the following:

- Typescript provides a comprehensive set of operators.
- The + and += operators can be used to create and concatenate strings.
- Use decision-making constructs
- Perform loop operations
- Write `switch` statements