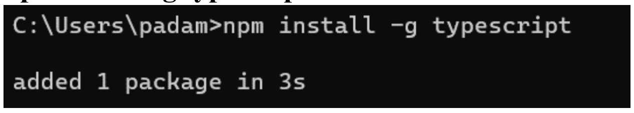
Typescript Operator

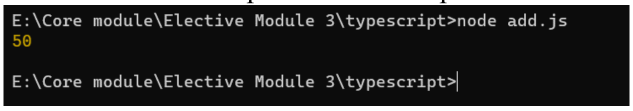
An Operator is a symbol which operates on a value or data. It represents a specific action on working with data. The data on which operators operates is called operand. It can be used with one or more than one values to produce a single value. All of the standard JavaScript operators are available with the TypeScript program.

In TypeScript, an operator can be classified into the following ways:

- Arithmetic operators
- Comparison (Relational) operators
- Logical operators
- Bitwise operators
- Assignment operators
- Ternary/conditional operator
- Concatenation operator
- Type Operator

Before performing any operation, we need to install TypeScript on our physical machine. Follow these steps for installation.

1. Go to the cmd and check the version of node and npm.
   npm -v(this command for npm version).
   node -v(this command for node version).
2. Now install typescript.
   npm install -g typescript.
   

   ```
   C:\Users\padam>npm install -g typescript

   added 1 package in 3s
   ```
3. Then we need to create a new file with **.ts**(it's file extension for Typescript files).
4. I have created a new file named " **arithmetic.ts**" and I have performed all operators within this file.
5. After all steps we need to change file into the .js extension file for that use this command.
   tsc arithmetic.ts(this command creates a new file with same name but **.js** extension).
6. Then we need to show file output.
   node arithmetic.js(this command uses to sow file output).
   We can run .ts file also in a latest update of tsc.
   like this it's an example of addition operator.
   

   ```
   E:\Core module\Elective Module 3\typescript>node add.js
   50

   E:\Core module\Elective Module 3\typescript>
   ```

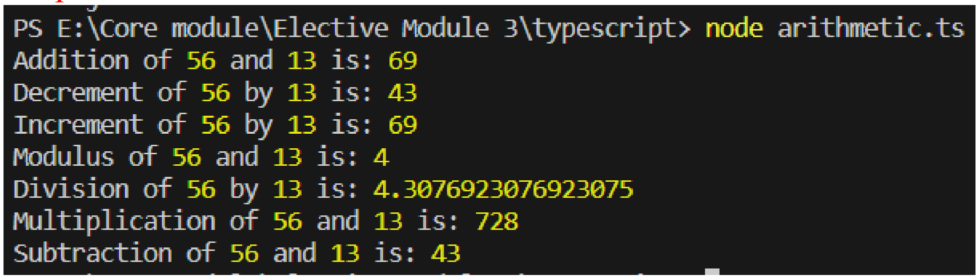
Arithmetic Operators

Arithmetic operators take numeric values as their operands, performs an action, and then returns a single numeric value. The most common arithmetic operators are addition(+), subtraction(-), multiplication(*), and division(/).

Input:

```
let a = 56;
let b = 13;

// Addition
console.log('Addition of', a, 'and', b, 'is:', a + b);

// Decrement
console.log('Decrement of', a, 'by', b, 'is:', a - b);

// Increment
console.log('Increment of', a, 'by', b, 'is:', a + b);

// Modulus
console.log('Modulus of', a, 'and', b, 'is:', a % b);

// Division
console.log('Division of', a, 'by', b, 'is:', a / b);

// Multiplication
console.log('Multiplication of', a, 'and', b, 'is:', a * b);

// Subtraction
console.log('Subtraction of', a, 'and', b, 'is:', a - b);
```

Output:



```
PS E:\Core module\Elective Module 3\typescript> node arithmetic.ts
Addition of 56 and 13 is: 69
Decrement of 56 by 13 is: 43
Increment of 56 by 13 is: 69
Modulus of 56 and 13 is: 4
Division of 56 by 13 is: 4.3076923076923075
Multiplication of 56 and 13 is: 728
Subtraction of 56 and 13 is: 43
```

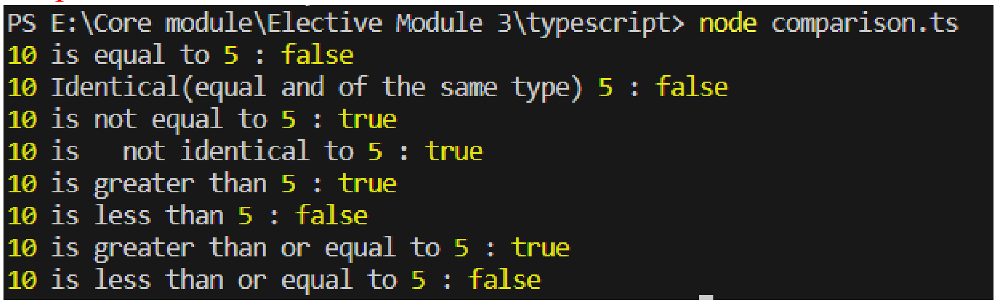
Comparison (Relational) Operators

The comparison operators are used to compares the two operands. These operators return a Boolean value true or false. The important comparison operators are given below.

Input:

```
let x = 10;
let y = 5;

// Equal to
console.log(x, 'is equal to', y, ':', x === y);

// Equal to
console.log(x, 'Identical(equal and of the same type)', y, ':', x === y);

// Not equal to
console.log(x, 'is not equal to', y, ':', x != y);

// Not equal to
console.log(x, 'is            not identical to', y, ':', x !== y);

// Greater than
console.log(x, 'is greater than', y, ':', x > y);

// Less than
console.log(x, 'is less than', y, ':', x < y);

// Greater than or equal to
console.log(x, 'is greater than or equal to', y, ':', x >= y);

// Less than or equal to
console.log(x, 'is less than or equal to', y, ':', x <= y);
```

Output:



```
PS E:\Core module\Elective Module 3\typescript> node comparison.ts
10 is equal to 5 : false
10 Identical(equal and of the same type) 5 : false
10 is not equal to 5 : true
10 is   not identical to 5 : true
10 is greater than 5 : true
10 is less than 5 : false
10 is greater than or equal to 5 : true
10 is less than or equal to 5 : false
```

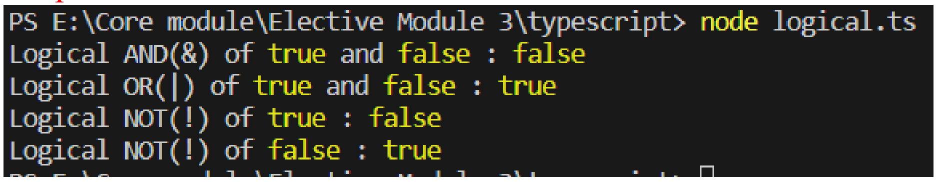
Logical Operators

Logical operators are used for combining two or more condition into a single expression and return the Boolean result true or false. The Logical operators are given below.

Input:

```
let x = true;
let y = false;

// Logical AND
console.log('Logical AND(&) of', x, 'and', y, ':', x && y);

// Logical OR
console.log('Logical OR(|) of', x, 'and', y, ':', x || y);

// Logical NOT
console.log('Logical NOT(!) of', x, ':', !x);
console.log('Logical NOT(!) of', y, ':', !y);
```

Output:



```
PS E:\Core module\Elective Module 3\typescript> node logical.ts
Logical AND(&) of true and false : false
Logical OR(|) of true and false : true
Logical NOT(!) of true : false
Logical NOT(!) of false : true
```

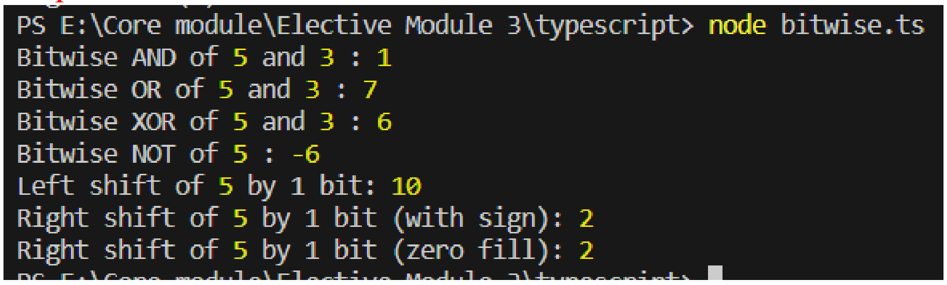
Bitwise Operators

The bitwise operators perform the bitwise operations on operands. The bitwise operators are as follows.

Input:

```
let a = 5;  //  binary: 0101
let b = 3;  //  binary: 0011

// Bitwise AND
console.log('Bitwise AND of', a, 'and', b, ':', a & b);  // Result: 1 (binary: 0001)

// Bitwise OR
console.log('Bitwise OR of', a, 'and', b, ':', a | b);   // Result: 7 (binary: 0111)

// Bitwise XOR (exclusive OR)
console.log('Bitwise XOR of', a, 'and', b, ':', a ^ b);   // Result: 6 (binary: 0110)

// Bitwise NOT
console.log('Bitwise NOT of', a, ':', ~a);               // Result: -6 (in two's complement form)

// Left shift
console.log('Left shift of', a, 'by 1 bit:', a << 1);    // Result: 10 (binary: 1010)

// Right shift (with sign)
console.log('Right shift of', a, 'by 1 bit (with sign):', a >> 1);  // Result: 2 (binary: 0010)

// Right shift (zero fill)
console.log('Right shift of', a, 'by 1 bit (zero fill):', a >>> 1);  // Result: 2 (binary: 0010)
```

Output:



```
PS E:\Core module\Elective Module 3\typescript> node bitwise.ts
Bitwise AND of 5 and 3 : 1
Bitwise OR of 5 and 3 : 7
Bitwise XOR of 5 and 3 : 6
Bitwise NOT of 5 : -6
Left shift of 5 by 1 bit: 10
Right shift of 5 by 1 bit (with sign): 2
Right shift of 5 by 1 bit (zero fill): 2
PS E:\Core module\Elective Module 3\typescript>
```

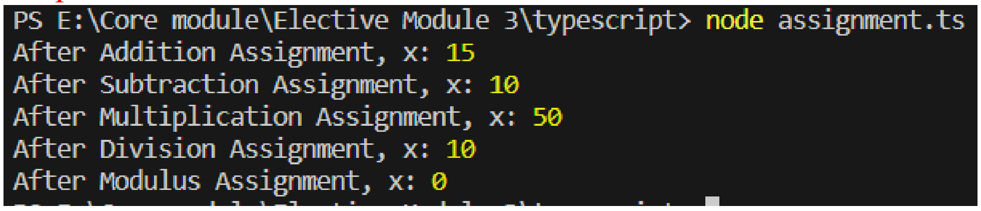
Assignment Operators

Assignment operators are used to assign a value to the variable. The left side of the assignment operator is called a variable, and the right side of the assignment operator is called a value. The data-type of the variable and value must be the same otherwise the compiler will throw an error. The assignment operators are as follows.

Input:

```
let x = 10;
let y = 5;

// Addition Assignment
x += y; // Equivalent to: x = x + y; (x becomes 15)
console.log('After Addition Assignment, x:', x);

// Subtraction Assignment
x -= y; // Equivalent to: x = x - y; (x becomes 10)
console.log('After Subtraction Assignment, x:', x);

// Multiplication Assignment
x *= y; // Equivalent to: x = x * y; (x becomes 50)
console.log('After Multiplication Assignment, x:', x);

// Division Assignment
x /= y; // Equivalent to: x = x / y; (x becomes 10)
console.log('After Division Assignment, x:', x);

// Modulus Assignment
x %= y; // Equivalent to: x = x % y; (x becomes 0)
console.log('After Modulus Assignment, x:', x);
```

Output:



```
PS E:\Core module\Elective Module 3\typescript> node assignment.ts
After Addition Assignment, x: 15
After Subtraction Assignment, x: 10
After Multiplication Assignment, x: 50
After Division Assignment, x: 10
After Modulus Assignment, x: 0
```

Ternary/Conditional Operator

The conditional operator takes three operands and returns a Boolean value based on the condition, whether it is true or false. Its working is similar to an if-else statement. The conditional operator has right-to-left associativity. The syntax of a conditional operator is given below.

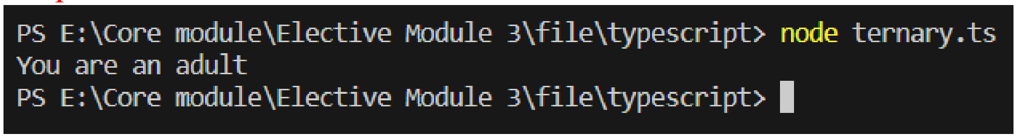
expression ? expression-1 : expression-2;

- **expression:** It refers to the conditional expression.
- **expression-1:** If the condition is true, expression-1 will be returned.
- **expression-2:** If the condition is false, expression-2 will be returned.

Input:

```
let age = 20;

// Ternary operator to check if age is greater than or equal to 18
let message = age >= 18 ? 'You are an adult' : 'You are a minor';

console.log(message);
```

Output:



```
PS E:\Core module\Elective Module 3\file\typescript> node ternary.ts
You are an adult
PS E:\Core module\Elective Module 3\file\typescript>
```

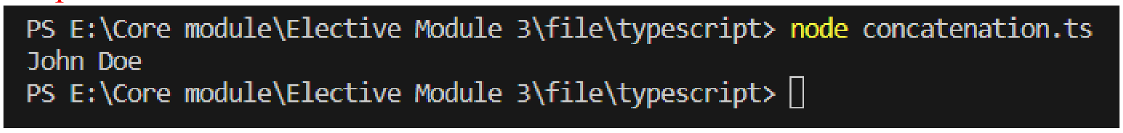
Concatenation Operator

The concatenation (+) operator is an operator which is used to append the two strings. In concatenation operation, we cannot add a space between the strings. We can concatenate multiple strings in a single statement. The following example helps us to understand the concatenation operator in TypeScript.

Input:

```
let firstName = "John";
let lastName = "Doe";

// Concatenation using the + operator
let fullName = firstName + " " + lastName;

console.log(fullName);  // Output: "John Doe"
```

Output:



```
PS E:\Core module\Elective Module 3\file\typescript> node concatenation.ts
John Doe
PS E:\Core module\Elective Module 3\file\typescript>
```

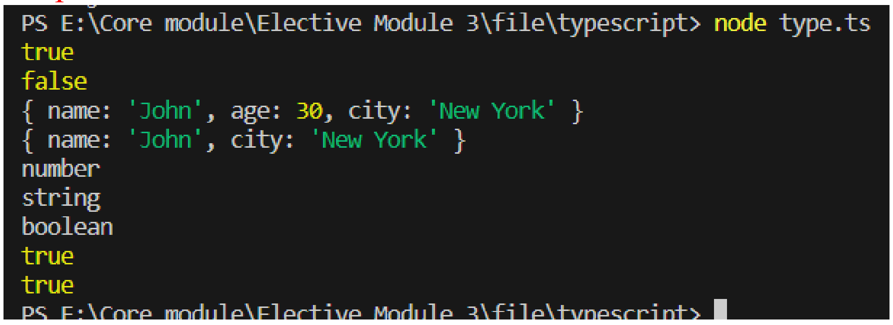
Type Operators

There are a collection of operators available which can assist you when working with objects in TypeScript. Operators such as typeof, instanceof, in, and delete are the examples of Type operator. The detail explanation of these operators is given below.

Input:

```
// Creating an object
let person = {
    name: "John",
    age: 30,
    city: "New York"
  };

  // Check if a property exists in an object using the 'in' operator
  console.log("name" in person);  // Output: true
  console.log("gender" in person);  // Output: false

  // Deleting a property from an object using the 'delete' operator
  console.log(person);  // Before deletion
  delete person.age;
  console.log(person);  // After deletion

  // Using the 'typeof' operator to check the type of a variable
  let variable1 = 42;
  let variable2 = "Hello";
  let variable3 = true;

  console.log(typeof variable1);  // Output: "number"
  console.log(typeof variable2);  // Output: "string"
  console.log(typeof variable3);  // Output: "boolean"

  // Using the 'instanceof' operator to check if an object is an instance of a class
  class Car {
    constructor(make) {
      this.make = make;
    }
  }
  let myCar = new Car("Toyota");
  console.log(myCar instanceof Car);  // Output: true
  console.log(myCar instanceof Object);  // Output: true (since all objects are instances of Object)
```

Output:



```
PS E:\Core module\Elective Module 3\file\typescript> node type.ts
true
false
{ name: 'John', age: 30, city: 'New York' }
{ name: 'John', city: 'New York' }
number
string
boolean
true
true
PS E:\Core module\Elective Module 3\file\typescript>
```