

Day 1: Arithmetic Operators



by [AvmnuSng](#)

Problem

Submissions

Leaderboard

Discussions

Editorial

Tutorial

Arithmetic Operators

Operator Types

Unary

A *unary* operator requires a single operand, either before or after the operator, following

Table Of Contents

[Operator Types](#)

[Arithmetic Operators](#)

We use cookies to ensure you have the best browsing experience on our website. Please read our [cookie policy](#) for more information about how we use cookies.

OK

```
operand operator
operator operand
```

For example, in the expression `a++`, `++` is a unary operator.

Binary

A *binary* operator requires two operands, one before the operator and one after the operator, following this format:

```
operand1 operator operand2
```

For example, in the expression `a + b = c`, `+` is a binary operator.

Ternary

There is one *ternary* operator, the conditional operator. For example, in the expression `a ? b : c`, the use of `?` and `:` in this manner constitutes the ternary operator. We'll discuss this operator more in the *Conditional Statements* tutorial.

Arithmetic Operators

An arithmetic operator takes numeric values (either literals or variables) as its operands and returns a single numeric value. The standard arithmetic operators are addition (`+`), subtraction (`-`), multiplication (`*`), and division (`/`). Other arithmetic operators are

1. Addition (+)

We use this operator in the form `operand1 + operand_2`. For example:

```
2 + 3 // evaluates to 5
4 + 10 // evaluates to 14
```

2. Subtraction (-)

We use this operator in the form `operand1 - operand2`. For example:

```
3 - 2 // evaluates to 1
4 - 10 // evaluates to -6
```

3. Multiplication (*)

We use this operator in the form `operand1 * operand2`. For example:

```
3 * 2 // evaluates to 6
4 * 10 // evaluates to 40
```

4. Division (/)

We use this operator in the form `operand1 / operand2`. For example:

```
6 / 3 // evaluates to 2
```

5. Remainder (%)

We use this operator in the form `operand1 % operand2`. For example:

```
6 % 3 // evaluates to 0
3 % 2 // evaluates to 1
4 % 10 // evaluates to 4
```

6. Exponentiation (**)

We use this operator in the form `operand1 ** operand2`. This operator is a part of ECMAScript2016 feature set. For example:

```
2 ** 3 // evaluates to 8
3 ** 2 // evaluates to 9
5 ** 4 // evaluates to 625
```

7. Unary Negation (-)

We use this operator in the form `-operand`. For example:

```
-4 // evaluates to -4
-(-5) // evaluates to 5 (not --5)
```

8. Unary Plus (+)

We use this operator in the form `+operand`. For example:

```
+4 // evaluates to 4
+(-4) // evaluates to -4
```

9. Increment (++)

We use this operator in the prefix and postfix forms, forms `++operand` and `operand++`. The prefix form, `++operand`, increments the operand by **1** and then returns the value of the operand. The postfix form, `operand++`, returns the value of the operand and *then* increments the operand's value by **1**. For example:

EXAMPLE

```
process.stdin.on('data', function (data) {
  main(+data);
});
/**** Ignore above this line. ****/
function main(input) {
  var a = input;
  // Print the value of 'a' and the preincremented value of 'a':
  console.log("a(" + a + "), ++a(" + ++a + ")");
  // Assign the current value of 'a' to 'b' and then postincrement
  'a':
  var b = a++;
  // Print the values of 'a' once and 'b' twice, then postincrement
  the 2nd 'b':
  console.log("a(" + a + "), b(" + b + "), b++(" + b++ + ")");
  // Print the final values of 'a' and 'b':
  console.log("a(" + a + "), b(" + b + ")");
}
```

Input

4

Run

Output

Solution

The code above produces this output:

```
a(4), ++a(5)  
a(6), b(5), b++(5)  
a(6), b(6)
```

10. Decrement (--)

We use this operator in the prefix and postfix forms, forms `--operand` and `operand--`. The prefix form, `--operand`, decrements the operand by **1** and then returns the value of the operand. The postfix form, `operand--`, returns the value of the operand and *then* decrements the operand's value by **1**. For example:

EXAMPLE

```
process.stdin.on('data', function (data) {
    main(+data);
});
/**** Ignore above this line. ****/
function main(input) {
    var a = input;
    // Print the value of 'a' and the decremented value of 'a':
    console.log("a(" + a + "), --a(" + --a + ")");
    // Assign the current value of 'a' to 'b' and then postdecrement
    'a':
    var b = a--;
    // Print the values of 'a' once and 'b' twice, then postdecrement
    the 2nd 'b':
    console.log("a(" + a + "), b(" + b + "), b--(" + b-- + ")");
    // Print the final values of 'a' and 'b':
    console.log("a(" + a + "), b(" + b + ")");
}
```

Input

4

Run

Output

Solution

The code above produces this output:

```
a(4), --a(3)
a(2), b(3), b--(3)
a(2), b(2)
```

-

RECOMMENDED ARTICLE

-

RECOMMENDED ARTICLE

[Contest Calendar](#) | [Interview Prep](#) | [Blog](#) | [Scoring](#) | [Environment](#) | [FAQ](#) | [About Us](#) | [Support](#) | [Careers](#) | [Terms Of Service](#) | [Privacy Policy](#) | [Request a Feature](#)