6.2 Arithmetic

Arithmetic operators

An ***expression*** is a combination of items like variables, numbers, operators, and parentheses, that evaluates to a value like 2 * (x + 1). Expressions are commonly used on the right side of an assignment statement, as in `y = 2 * (x + 1)`.

An ***arithmetic operator*** is used in an expression to perform an arithmetic computation. Ex: The arithmetic operator for addition is +. JavaScript arithmetic operators are summarized in the table below.

Table 6.2.1: JavaScript arithmetic operators.

Arithmetic operator	Description	Example
+	Add	`// x = 3` `x = 1 + 2;`
-	Subtract	`// x = 1` `x = 2 - 1;`
*	Multiply	`// x = 6` `x = 2 * 3;`
/	Divide	`// x = 0.5` `x = 1 / 2;`
%	Modulus (remainder)	`// x = 0` `x = 4 % 2;`
**	Exponentiation	`// x = 2 * 2 * 2 = 8` `x = 2 ** 3;`
++	Increment	`// Same as x = x + 1` `x++;`

Arithmetic operator	Description	Example
`--`	Decrement	`// Same as x = x - 1` `x--;`

Feedback?

Expressions are computed using the same rules as basic arithmetic. Expressions in parentheses `()` have highest precedence, followed by exponentiation (`**`). Multiplication (`*`), division (`/`), and modulus (`%`) have precedence over addition (+) and subtraction (–). Ex: The expression `7 + 3 * 2` = 7 + 6 = 13 because `*` has precedence over +, but `(7 + 3) * 2` = 10 * 2 = 20 because `()` has precedence over `*`.

PARTICIPATION ACTIVITY 6.2.1: Arithmetic practice.

What is `points` at the end of each code segment?

1)
```
points = 3 + 5 * 2;
```
13

Check Show answer

Correct

13

3 + 5 * 2 = 3 + 10 = 13

2)
```
points = 4;
points = (3 + points) % 5;
```
2

Check Show answer

Correct

2

(3 + 4) % 5 = 7 % 5 = 2

3)
```
scale = 5;
points = 3 ** 2 * scale;
```
45

Check Show answer

Correct

45

3 ** 2 * 5 = 9 * 5 = 45. The `**` operator has higher precedence than the basic arithmetic operators like + and `*`.

4)
```
points = 10;
points--;
```
9

Correct

9

Check Show answer

`points--` is the same as `points = points - 1`, so `points` is assigned 10 - 1 = 9.

5)

```
points = 6;
points++;
```

7

Check Show answer

Correct

7

`points++` is the same as `points = points + 1`, so `points` is assigned 6 + 1 = 7.

Feedback?

Compound assignment operators

A ***compound assignment operator*** combines an assignment statement with an arithmetic operation. Common JavaScript compound assignment operators are summarized in the table below.

Table 6.2.2: Compound assignment operators.

Assignment operator	Description	Example
+=	Add to	`// Same as x = x + 2` `x += 2;`
-=	Subtract from	`// Same as x = x - 2` `x -= 2;`
*=	Multiply by	`// Same as x = x * 3` `x *= 3;`
/=	Divide by	`// Same as x = x / 3` `x /= 3;`
%=	Mod by	`// Same as x = x % 4` `x %= 4;`

Feedback?

PARTICIPATION ACTIVITY 6.2.2: Practice with compound assignment operators.

1) What compound assignment operator makes `points` become 2.5?

```
points = 5;
points ___ 2;
```

/=

Check Show answer

Correct

/=

points /= 2 is the same as `points = points / 2`, so `points` is assigned 5 / 2 = 2.5.

2) What is `points`?

```
points = 2;
points *= 3 + 1;
```

8

Check Show answer

Correct

8

points *= 3 + 1 is the same as `points = points * (3 + 1)`, so `points` is assigned 2 * (3 + 1) = 2 * 4 = 8.

3) What is `points`?

```
points = 4;
points %= 2;
```

0

Check Show answer

Correct

0

points %= 2 is the same as `points = points % 2`, so `points` is assigned 4 % 2 = 0.

Feedback?

Arithmetic with numbers and strings

The + operator is also the string concatenation operator. ***String concatenation*** appends one string after the end of another string, forming a single string. Ex: `"back" + "pack"` is `"backpack"`.

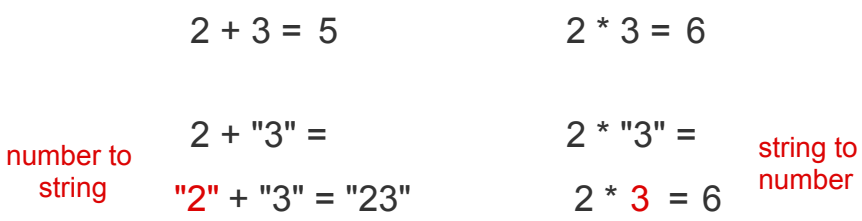
The JavaScript interpreter determines if + means "add" or "concatenate" based on the operands on either side of the operator. An ***operand*** is the value or values that an operator works on, like the number 2 or variable x.

- If both operands are numbers, + performs addition. Ex: 2 + 3 = 5.
- If both operands are strings, + performs string concatenation. Ex: "2" + "3" = "23".
- If one operand is a number and the other a string, + performs string concatenation. The number is converted into a string, and the two strings are concatenated into a single string. Ex: "2" + 3 = "2" + "3" = "23".

For all other arithmetic operators, combining a number and a string in an arithmetic expression converts the string operand to a number and then performs the arithmetic operation. Ex: "2" * 3 = 2 * 3 = 6.

PARTICIPATION ACTIVITY 6.2.3: Type conversion in arithmetic operations.

Start 2x speed



Captions

1. number + number = number
2. number + string = string
3. number * number = number
4. number * string = number

Feedback?

The JavaScript functions ***parseInt()*** and ***parseFloat()*** convert strings into numbers. Ex: `parseInt("5") + 2 = 5 + 2 = 7`, and `parseFloat("2.4") + 6 = 2.4 + 6 = 8.4`.

If `parseInt()` or `parseFloat()` are given a non-number to parse, the functions return `NaN`. ***NaN*** is a JavaScript value that means Not a Number. Ex: `parseInt("dog")` is `NaN`.

The JavaScript function ***isNaN()*** returns `true` if the argument is not a number, `false` otherwise. When the `isNaN()` argument is non-numeric, the function attempts to convert the argument into a number. Ex: `isNaN("dog")` is `true` because the non-numeric value "dog" cannot be converted into a number. But `isNaN("123")` is `false` because "123" can be converted into the number 123.

PARTICIPATION ACTIVITY 6.2.4: Arithmetic practice with numbers and strings.

What is `secretCode` at the end of each code segment? Type "quotes" around strings. If not a number, type NaN.

1)
```
secretCode = 10 +
"ten";
```

"10ten"

Correct

"10ten"

Check Show answer

+ performs string concatenation when one operand is a number and one is a string. 10 + "ten" = "10" + "ten" = "10ten"

2) `secretCode = "3" / "6";`

0.5

Check Show answer

Correct

0.5

Arithmetic with / converts "3" and "6" into numbers before dividing. "3" / "6" = 3 / 6 = 0.5

3) `secretCode = "3" + 5 * 2;`

"310"

Check Show answer

Correct

"310"

"3" + 5 * 2 = "3" + 10 = "3" + "10" = "310". * has precedence, so 5 * 2 is evaluated first. Then, "3" + 10 converts 10 to a string and concatenates "10" to the string "3", resulting in "310".

4) `secretCode = parseFloat("3.2") + parseInt("2.7");`

5.2

Check Show answer

Correct

5.2

parseInt() ignores anything after the decimal place. parseFloat("3.2") + parseInt("2.7") = 3.2 + 2 = 5.2

5) `secretCode = 3 + parseInt("pig");`

NaN

Check Show answer

Correct

NaN

Arithmetic performed with `NaN` results in `NaN`. 3 + parseInt("pig") = 3 + NaN = NaN

6) `// true = 1, false = 0`
`secretCode = 2 + isNaN("oink") + isNaN("5");`

3

Check Show answer

Correct

3

`isNaN("oink")` = true = 1 in an arithmetic computation, and `isNaN("5")` = false = 0. So 2 + isNaN("oink") + isNaN("5") = 2 + true + false = 2 + 1 + 0 = 3

Feedback?

CHALLENGE ACTIVITY 6.2.1: Arithmetic operators.

530096.4000608.qx3zqy7

Start

Write a statement that assigns finalValue with the sum of value1 and value2. Ex: If value1 is 6 and value2 is 2, finalValue is 8. Note: Your code will be tested with more values for value1 and value2.

```
1 let value1 = 6;
2 let value2 = 2;
3
4 /* Modify the following line */
5 let finalValue = 0;
6
```

1 2 3 4

Check Next

View your last submission

Feedback?

Exploring further:

- Arithmetic operators from MDN
- MDN documentation for parseInt(), parseFloat(), and isNaN()

How was this section? Provide section feedback