Teaching guide: arithmetic operations in a programming language

This resource will help with understanding Arithmetic operations in a programming language. It supports teaching of Section 3.2.3 of our GCSE Computer Science specification (8525). This resource focuses on the following learning outcomes:

- Evaluate expressions using numbers
- Recognise the difference between integer and real division.

Standard arithmetic operations

Students will be familiar with the most common numerical operations in programming languages: addition, subtraction and multiplication are unchanged from the ways learned in primary school. For example:

`34 + 4` evaluates to `38`

`12 - 10` evaluates to `2`, and

`3 * 4` evaluates to `12`

In the last example, instead of using the `×` symbol for multiplication (which can be confused with the character `'x'`) programming languages normally use the `*` symbol instead. In all other respects these three operations are the same as in normal mathematics.

It is possible to build more complex expressions, for example:

`3 + 54 - 7` evaluates to `50`, and

`2 * 3 * 4` evaluates to `24`

Programming languages use their own version of BIDMAS (Brackets, Indices, Division, Multiplication, Addition and Subtraction) to give the standard order of precedence in arithmetic (the order in which parts of a complex expression are evaluated).

However, different programming languages have different rules and you should not assume, for example, that division would be carried out before multiplication. The use of brackets (parentheses) eliminates any ambiguity.

`(3 + 4) * (4 - 2)` clearly evaluates to `14`

For division the / symbol is used instead of the ÷ symbol as the latter is not on a standard keyboard. This denotes real division (meaning the value of the expression is a real number), and so

`5 / 2` evaluates to `2.5`
`12 / 4` evaluates to `3` (alternatively `3.0` - integers can be considered as special kinds of real numbers, but the opposite is not true; `4`, an integer, is the same number as the real `4.0`. The real number `4.3` has no integer equivalent though.)
`25 / 4` evaluates to `6.25`

Integer division and remainders

There is another kind of division used in programming. Using the last example, we would say that `25` divided by `4` gives `6` remainder `1`. Using more formal language we would say:

`25` divided by `4` evaluates to the quotient `6` with a remainder of `1`

There are two different operations being performed here:

1. the integer division operation (**`DIV`**) calculates the quotient and
2. the modulus operation (**`MOD`**) calculates the remainder.

So,

`25` **`DIV`** `4` evaluates to `6`
`25` **`MOD`** `4` evaluates to `1`

Both expressions result in an integer value.

In general, `m` **`MOD`** `n` will evaluate to an integer between `0` and `n-1` where `m` and `n` are any integer. Further examples are:

`14` **`DIV`** `3` evaluates to `4`, and
`14` **`MOD`** `3` evaluates to `2`

`17` **`DIV`** `5` evaluates to `3`, and
`17` **`MOD`** `5` evaluates to `2`

There are many applications where the modulus operator is useful, such as working out if a number is odd or even.

If you divide any integer by `2` it will result in a remainder of `0` if it is an even number and a remainder of `1` if it is an odd number – this is a quick way to check a number's parity (odd or even).

Another example is putting people into groups. If every student in a class is given a number between `1` and `30` and they need to be in four groups you could ask them all to evaluate their number modulo `4` and they would each have the value `0`, `1`, `2` or `3`.