Properties of assert

This lesson explains different properties associated with assert in detail.

WE'LL COVER THE FOLLOWING

Using `assert` checks

Use `assert` checks even if absolutely true. A large set of program errors are caused by assumptions that are thought to be absolutely true.
For that reason, take advantage of `assert` checks even if they feel unnecessary. Let's look at the following function that returns the days of months in a given year:

```
int[] monthDays(int year) {
    int[] days = [
       31, februaryDays(year),
       31, 30, 31, 30, 31, 31, 30, 31, 30, 31
    ];

    assert((sum(days) == 365) ||
           (sum(days) == 366));

    return days;
}
```

That `assert` check may be seen as unnecessary because the function would naturally return either 365 or 366. However, those checks are guarding against potential mistakes even in the `februaryDays()` function. For example, the program would be terminated if `februaryDays()` returned 30.
Another seemingly unnecessary check can ensure that the length of the slice

Another seemingly unnecessary check can ensure that the length of the slice would always be 12:

```
assert(days.length == 12);
```

That way, deleting or adding elements to the slice unintentionally would also be caught. Such checks are important tools for program correctness.

`assert` is also the fundamental tool that is used in unit testing and contract programming, both of which will be covered in later chapters.

`assert` check's value

We have seen that expressions produce values or make side effects. `assert` checks do not have values nor should they have any side effects.
The D language requires that the evaluation of the logical expression must not have any side effects. `assert` must remain as a passive observer of the program state.

Disabling `assert` checks

Since `assert` is about program correctness, `assert` checks can be seen as unnecessary once the program has been tested sufficiently. Furthermore, since `assert` checks neither produce values nor have side effects, removing them from the program should not make any difference.
The compiler switch `-release` causes the `assert` checks to be ignored as if they have never been included in the program:

```
$ dmd deneme.d -release
```

This would allow programs to run faster by not evaluating potentially slow logical expressions of the `assert` checks.
As an exception, the `assert` checks that have the literal false (or 0) as the logical expression are not disabled even when the program is compiled with -release. This is because `assert(false)` is for ensuring that a block of code is never reached, and that should be prevented even for the -release compilations.

In the next lesson, we will look at `enforce` and the use of `assert`/`enforce`.