enforce and the Use of assert-enforce

This lesson explains the use of enforce statements and assert in D language.

WE'LL COVER THE FOLLOWING

`enforce` for throwing exceptions

Not every unexpected situation is an indication of a program error. Programs may also experience unexpected inputs and unexpected environmental states. For example, the data that is entered by the user should not be validated by an `assert` check because invalid data has nothing to do with the correctness of the program itself. In such cases, it is appropriate to throw exceptions like we have been doing in previous programs.

`std.exception.enforce` is a convenient way of throwing exceptions. For example, let's assume that an exception must be thrown when a specific condition is not met:

```
if (count < 3) {
    throw new Exception("Must be at least 3.");
}
```

`enforce()` is a wrapper around the condition check and the `throw` statement. The following is the equivalent of the previous code:

```
import std.exception;
// ...
    enforce(count >= 3, "Must be at least 3.");
```

> **Note:** Observe how the logical expression is negated compared to the if

> **Note:** Observe how the logical expression is negated compared to the if statement. It now spells out what is being enforced.

How to use

`assert` is for catching programmer errors. The conditions that `assert` guards against in the `monthDays()` function and the `menuTitle` variable above are all about mistakes.
Sometimes it is difficult to decide whether to rely on an `assert` check or to throw an exception. The decision should be based on whether the unexpected situation is due to a problem with how the program has been coded.

Otherwise, the program must throw an exception when it is not possible to accomplish a task. `enforce()` is expressive and convenient when throwing exceptions.

Another point to consider is whether the unexpected situation can be remedied in some other way. If the program can not do anything special, such as displaying an error message about the problem with input data, then it is appropriate to throw an exception. That way, callers of the code that threw the exception can catch it to do something special to recover from the error condition.

In the next lesson, you will find a coding challenge to test the concepts covered in this chapter.