Test-Driven Development

This lesson explains a software development methodology that focuses on unit testing.

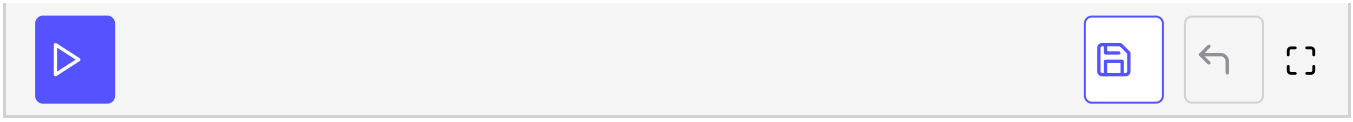
WE'LL COVER THE FOLLOWING

Test-driven development

Test-driven development (TDD) is a software development methodology that prescribes writing unit tests before implementing functionality. In TDD, the focus is on unit testing. Coding is a secondary activity that makes the tests pass.

In accordance with TDD, the `ordinal()` function below can first be implemented intentionally incorrectly:

```
import std.string;

string ordinal(size_t number) {
    return "";    // ← intentionally wrong
}

unittest {
    assert(ordinal(1) == "1st");
    assert(ordinal(2) == "2nd");
    assert(ordinal(3) == "3rd");
    assert(ordinal(10) == "10th");
}

void main() {
}
```

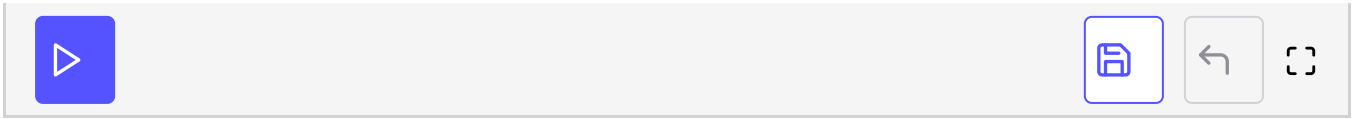


Unittest failure example

The function should be implemented only after seeing the failure and only to

make the tests pass. Here is just one implementation that passes the tests:

```
import std.string;

string ordinal(size_t number) {
    string suffix;

    switch (number) {
    case  1: suffix = "st"; break;
    case  2: suffix = "nd"; break;
    case  3: suffix = "rd"; break;
    default: suffix = "th"; break;
    }

    return format("%s%s", number, suffix);
}

unittest {
    assert(ordinal(1) == "1st");
    assert(ordinal(2) == "2nd");
    assert(ordinal(3) == "3rd");
    assert(ordinal(10) == "10th");
}

void main() {
}
```



Successful unittest example

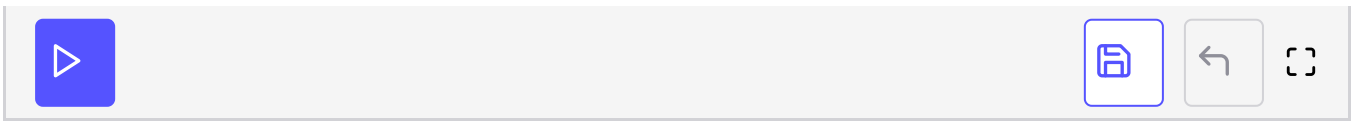
Unit tests before bug fixes

Unit tests are not a panacea; there will always be bugs. If a bug is discovered when actually running the program, it can be seen as an indication that the unit tests have been incomplete. For that reason, it is better to first write a unit test that reproduces the bug and only then fix the bug to pass the new test.

Let's have a look at the following function that returns the spelling of the ordinal form of a number specified as a `dstring`:

```
import std.exception;
import std.string;

dstring ordinalSpelled(dstring number) {
    enforce(number.length, "number cannot be empty");

    dstring[dstring] exceptions = [
        "one": "first", "two" : "second", "three" : "third",
        "five" : "fifth", "eight": "eighth", "nine" : "ninth",
        "twelve" : "twelfth"
```

```
    ];

    dstring result;

    if (number in exceptions) {
        result = exceptions[number];

    } else {
        result = number ~ "th";
    }

    return result;
}

unittest {
    assert(ordinalSpelled("one") == "first");
    assert(ordinalSpelled("two") == "second");
    assert(ordinalSpelled("three") == "third");
    assert(ordinalSpelled("ten") == "tenth");
}

void main() {
}
```

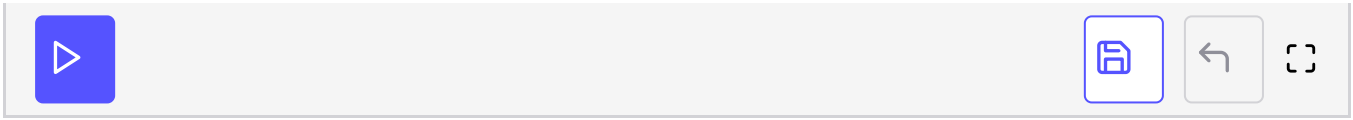


Function with a unittest still has a bug

The function takes care of exceptional spellings and even includes a unit test for that. Still, the function has a bug yet to be discovered:

```
import std.stdio;
import std.exception;
import std.string;

dstring ordinalSpelled(dstring number) {
    enforce(number.length, "number cannot be empty");

    dstring[dstring] exceptions = [
        "one": "first", "two" : "second", "three" : "third",
        "five" : "fifth", "eight": "eighth", "nine" : "ninth",
        "twelve" : "twelfth"
    ];

    dstring result;

    if (number in exceptions) {
        result = exceptions[number];

    } else {
        result = number ~ "th";
    }

    return result;
}
```

```
unittest {
    assert(ordinalSpelled("one") == "first");
    assert(ordinalSpelled("two") == "second");

    assert(ordinalSpelled("three") == "third");
    assert(ordinalSpelled("ten") == "tenth");

}

void main() {
    writefln("He came the %s in the race.", ordinalSpelled("twenty"));
}
```



Wrong output

The spelling error in the output of the program is due to a bug in `ordinalSpelled()`, which its unit tests fail to catch.
Although it is easy to see that the function does not produce the correct spelling for numbers that end with the letter y, TDD prescribes that first a unit test must be written to reproduce the bug before actually fixing it:

```
unittest {
// ...
    assert(ordinalSpelled("twenty") == "twentieth"); }
```

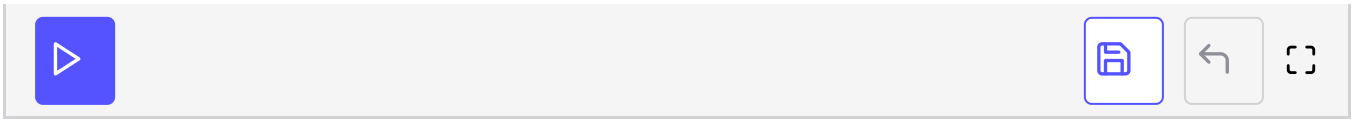
With that improvement to the tests, now the bug in the function is being caught during development:

```
core.exception.AssertError@deneme(3274338): unittest failure
```

The function should be fixed only then:

```
import std.stdio;
import std.exception;
import std.string;

dstring ordinalSpelled(dstring number) {
    enforce(number.length, "number cannot be empty");

    dstring[dstring] exceptions = [
        "one": "first", "two" : "second", "three" : "third",
        "five" : "fifth", "eight": "eighth", "nine" : "ninth",
        "twelve" : "twelfth"
    ];

    dstring result;

    if (number in exceptions) {
```

```
    if (number in exceptions) {
        result = exceptions[number];

    } else {
        if (number[$-1] == 'y') {
            result = number[0..$-1] ~ "ieth";

        } else {
            result = number ~ "th";
        }
    }

    return result;
}

unittest {
    assert(ordinalSpelled("one") == "first");
    assert(ordinalSpelled("two") == "second");
    assert(ordinalSpelled("three") == "third");
    assert(ordinalSpelled("ten") == "tenth");
    assert(ordinalSpelled("twenty") == "twentieth");

}

void main() {
    writefln("He came the %s in the race.", ordinalSpelled("twenty"));
}
```



Fixing bugs using TDD

The function above still does not work properly with two-digit numbers greater than twenty.

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