Using Table-Driven Tests

This lesson explains a method to organize the testing of Go programs via tables.

When writing tests, it is good practice to use an array to collect the test-inputs and the expected results together. Each entry in the table then becomes a complete test case with inputs and expected results, and sometimes with additional information such as a test name to make the test output more informative.

The actual test simply iterates through all table entries and performs the necessary tests for each entry. It could be abstracted in the following snippet:

```
var tests = []struct{ // Test table
  in string
  out string
}{
  {"in1", "exp1"},
  {"in2", "exp2"},
  {"in3", "exp3"},
...
}

func TestFunction(t *testing.T) {
  for i, tt := range tests {
    s := FuncToBeTested(tt.in)
    if s != tt.out {
      t.Errorf("%d. %q => %q, wanted: %q", i, tt.in, s, tt.out)
    }
  }
}
```

If a lot of test functions have to be written that way, it could be useful to write the actual test in a helper function `verify`:

```
func verify(t *testing.T, testnum int, testcase, input, output, expected s
tring) {
  if input != output {
```

```
  if input != output {

    t.Errorf("%d. %s with input = %s: output %s != %s", testnum, testcase,
 input,
output, expected)

  }
}
```

so that `TestFunction` becomes:

```
func TestFunction(t *testing.T) {
  for i, tt := range tests {
    s := FuncToBeTested(tt.in)

    verify(t, i, "FuncToBeTested: ", tt.in, s, tt.out)
  }
}
```

Now that you're familiar with error-handling and testing, let's see how to judge the performance of a Go program in the next lesson.