U5WEK2T4J : ```suite : Test
suite =
    describe "2048-elm"
        [test "moveLeftWithZero" <|
            _ ->
                let
                    expectedCases =
                        [([2, 0, 0, 2], [4, 0, 0, 0])
                        , ([2, 2, 0, 4], [4, 4, 0, 0])
                        , ([0, 0, 0, 4], [4, 0, 0, 0])
                        , ([0, 0, 2, 4], [2, 4, 0, 0])
                        , ([2, 4, 2, 4], [2, 4, 2, 4])
                        , ([2, 2, 2, 2], [4, 4, 0, 0])
                        ]

                    toTest =
                        List.map
                            (\expected ->
                                ((Tuple.first expected), (Main.moveLeftWithZero (Tuple.first expected)))
)
                            expectedCases
                in
                    Expect.equal expectedCases toTest
        ]
```

U5WEK2T4J : I tried with `Expect.all` but it does not seems to do what I want
U3SJEDR96 : seems like those should be multiple tests, actually?
U5WEK2T4J : for me it's the same test but with different data
U5WEK2T4J : so it's not worth having multiple tests
U5WEK2T4J : anyway it's working like that but I wanted to do something more "datadriven"
U48AEBJQ3 : <@U5WEK2T4J> I don't think this is how the `elm-test` authors want to construct tests. if you want to run a lot of cases through tests, it's generally better to write `Fuzz` tests.
If you are set on doing this, however, I think you will need to roll your own helper function to handle it.

U0JL9RPC4 : <@U48AEBJQ3> : right, but fuzzing supposes to know how to compute the result of the function you want to test and independantely of your original implementation
U0JL9RPC4 : with fuzzing, you have no control of the data your test generates, so you cannot challenge your test function result against a predefined computation
U48AEBJQ3 : I'm not going to argue the merits of `elm-test`, that is probably better left for <#C0CLGCMMF|testing>.
U5WEK2T4J : This another version thanks to <@U0JL9RPC4> ```
testMove when expected =
    test ("moveLeftWithZero" ++ (toString when)) &amp;lt;|
        \_ -&amp;gt;
            Expect.equal expected &amp;lt;| Main.moveLeftWithZero when


suite : Test
suite =
    describe "2048-elm"
        [ testMove [ 2, 0, 0, 2 ] [ 4, 0, 0, 0 ]
        , testMove [ 2, 2, 0, 4 ] [ 4, 4, 0, 0 ]
        , testMove [ 0, 0, 0, 4 ] [ 4, 0, 0, 0 ]
        , testMove [ 0, 0, 2, 4 ] [ 2, 4, 0, 0 ]
        , testMove [ 2, 4, 2, 4 ] [ 2, 4, 2, 4 ]
        , testMove [ 2, 2, 2, 2 ] [ 4, 4, 0, 0 ]
        ]
```

U0JL9RPC4 : <@U48AEBJQ3> don't get me wrong, I *love* the fuzzing in `elm-test`

U5WEK2T4J : It suits my need really well :slightly_smiling_face:
U3SJEDR96 : yeah, having them as separate tests like that makes sense to me - if only one case fails, one case fails
U5WEK2T4J : you're right <@U3SJEDR96> but I wanted to avoid having boilerplate code and with an helper function it's really readable and easy to add another test case
U0JL9RPC4 : I've already seen another strategy that involves indeed fuzzing in this case
U0JL9RPC4 : it means you have to write a naive and non optimized implementation of the function you want to test
U5WEK2T4J : anyway thank you folks helping me creating another 2048 clone in elm :smile:
U0JL9RPC4 : and run it against your "original" function for your fuzzed set
U0CLDU8UB : And if you want to use the list approach, that can be done easily with a helper too:```
testMove (when, expected) = ...

suite : Test
suite =
    describe "2048-elm" (List.map testMove expectedCases)
```

U3SJEDR96 : or even `(uncurry  testMove)` so you can keep your current implementation of it :wink:
U5WEK2T4J : thank you <@U0CLDU8UB> but how will it fill `when` and `expected` const with the tuple value ?
U3SJEDR96 : in ohanhi's setup, `testMove` takes a tuple and deconstructs it. with `uncurry`, that's handled by elm
U3SJEDR96 : well, handled by `uncurry` :stuck_out_tongue:
U5WEK2T4J : cool
U0JL9RPC4 : hihi, `uncurry` is also another door toward functional programming
U0CLDU8UB : So that syntax I wrote will do the same as your `Tuple.first` and `Tuple.second`, but already in the function definition.
U5WEK2T4J : I'll give it a try
U2LAL86AY : `datesInBetween: Date -&amp;gt; Date -&amp;gt; List Date` - does anyone know where to find a function that gives me all the dates between this 2 dates? ex: `datesInBetween "20 June" "25 June" -&amp;gt; [ 20 June, 21 June, 22 June, 23 June, 24 June, 25 June ] `
```