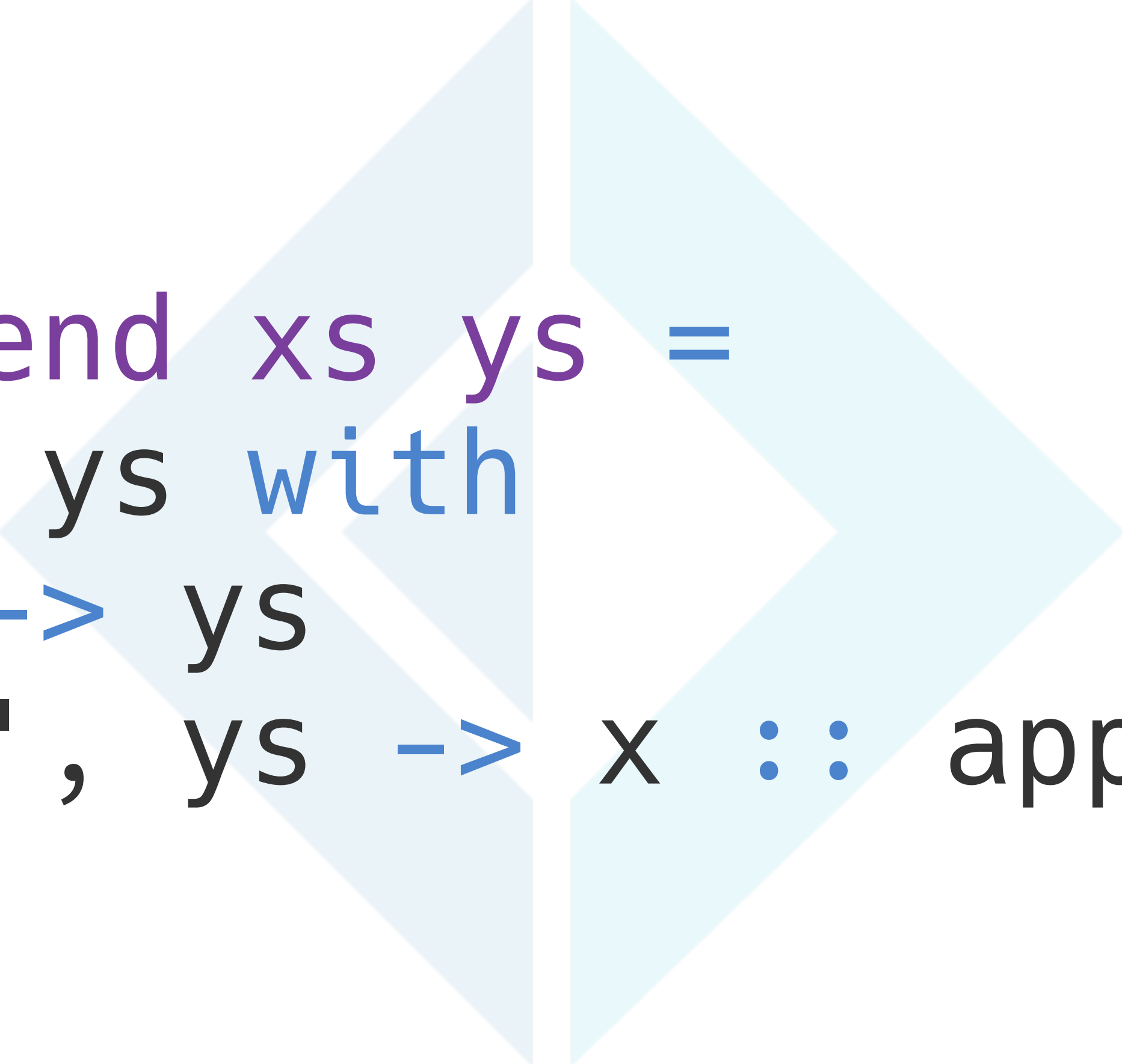




# 5 Levels of Testing

Mike Harris



```
let rec append xs ys =  
  match xs, ys with  
  | [], ys -> ys  
  | x :: xs', ys -> x :: append xs' ys
```

Manually test  
the code



# Level 1

Manual Testing

**Example**> **fsharp** **Tests.fs**

> **append** [] [1];;

val it : int list = [1]

> **append** [1] [];;

val it : int list = [1]

> **append** [1; 2] [3; 4];;

val it : int list = [1; 2; 3; 4]

Write hundreds of  
unit & integration  
tests for the code



# Level 2

Automated Testing



[<Fact>]

```
let ``Given empty list with list it must return list`` () =  
    Assert.Equal<Collections.Generic.IEnumerable<int>>(  
        [1], append [] [1])
```

[<Fact>]

```
let ``Given list with empty list it must return list`` () =  
    Assert.Equal<Collections.Generic.IEnumerable<int>>(  
        [1], append [1] [])
```

[<Fact>]

```
let ``Given two list it must concat them`` () =  
    Assert.Equal<Collections.Generic.IEnumerable<int>>(  
        [1; 2; 3; 4], append [1; 2] [3; 4])
```

Model the code  
and automatically  
generate exhaustive  
tests for every possible  
permutation of the code



# Level 3

## Property Testing

### [<Property>]

```
let ``Append with empty returns list``  
  (xs : int list) =  
    xs = append [] xs  
  && xs = append xs []
```

### [<Property>]

```
let ``Append will be length of both``  
  (xs : int list, ys : int list) =  
    List.length xs + List.length ys =  
      (append xs ys |> List.length)
```

### [<Property>]

```
let ``Append will contain members``  
  (x : int, ys : int list) =  
    append [x] ys |> List.contains x
```



Formally prove that  
it is mathematically  
impossible for the  
code to have bugs



# Level 4

Proof

**append** : **Vect** n a -> **Vect** m a -> **Vect** (m + n) a

append [] ys ?= ys

append (x :: xs) ys ?= x :: append xs ys

simple.append\_lemma\_2 = proof {

intros;

compute;

rewrite (plusSuccRightSucc m len);

trivial;

}

simple.append\_lemma\_1 = proof {

intros;

compute;

rewrite sym (plusZeroRightNeutral m);

exact value;

}

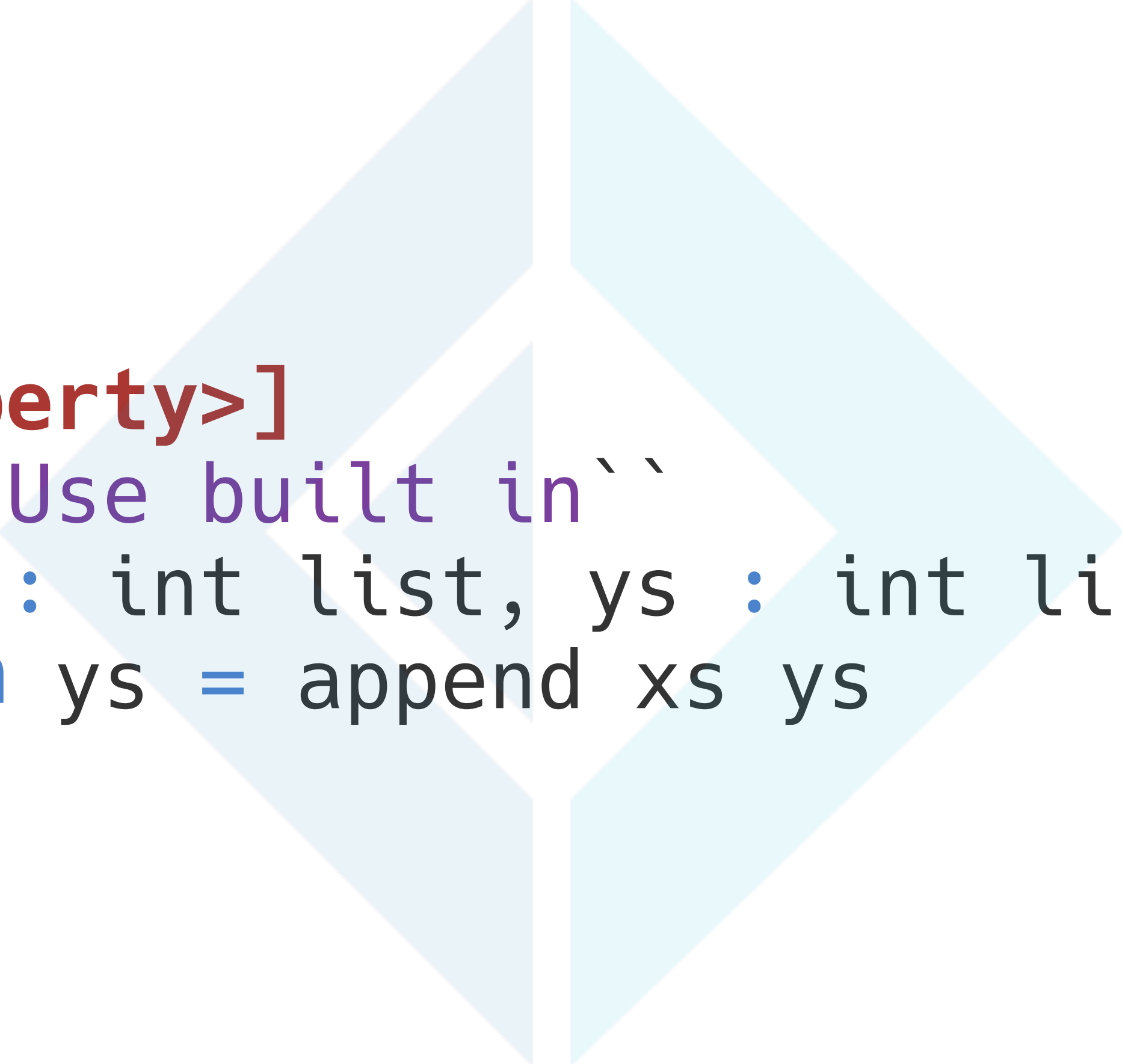


Delete the code



# Level 5

Delete the Code



```
[<Property>]
let ``Use built in``
  (xs : int list, ys : int list) =
    xs @ ys = append xs ys
```



```
let rec append xs ys =  
match xs, ys with  
| [], ys -> ys  
| x :: xs', ys -> x :: append xs' ys
```

`xs @ ys`

Manually test  
the code



Write hundreds of  
unit & integration  
tests for the code



Model the code  
and automatically  
generate exhaustive  
tests for every possible  
permutation of the code



Formally prove that  
it is mathematically  
impossible for the  
code to have bugs



Delete the code



**Less Obvious**

```
var total = 0.0;
foreach (var order in orders)
{
    if (order.Zip == 53202)
        total += order.Price * order.Quantity;
}
```



```
var total = orders
  .Where(order => order.Zip == 53202)
  .Select(order => order.Price * order.Quantity)
  .Aggregate(0.0, (sub, amount) => sub + amount);
```

# Take Aways

- Learn the frameworks you have available.
- Learn about different kinds of abstractions.
- Look for tools that already solve the problems you have.



# Thank you

Mike Harris

5 Levels of Testing  
<https://bit.ly/2sojN0b>

@MikeMKH



# Source

- [C# code] <https://github.com/MikeMKH/talks/blob/master/say-goodbye-to-the-for-loop-with-higher-order-functions/csharp/test/Example.cs>
- [F# logo] <https://fsharp.org/img/logo/fsharp512.png>
- [F# code] <https://github.com/MikeMKH/talks/blob/master/5-levels-of-testing/Example/Tests.fs>
- [Idris logo] <https://github.com/idris-lang/Idris-dev/blob/master/icons/text-x-idris.svg>
- [Idris code] <https://github.com/idris-lang/Idris-dev/blob/61cf812e97c0cf07a9596c1d36ab5a70eb5758b2/test/proof001/test029.idr#L21-L41>
- [Meme] <https://twitter.com/DavidKPiano/status/988479847352750080>