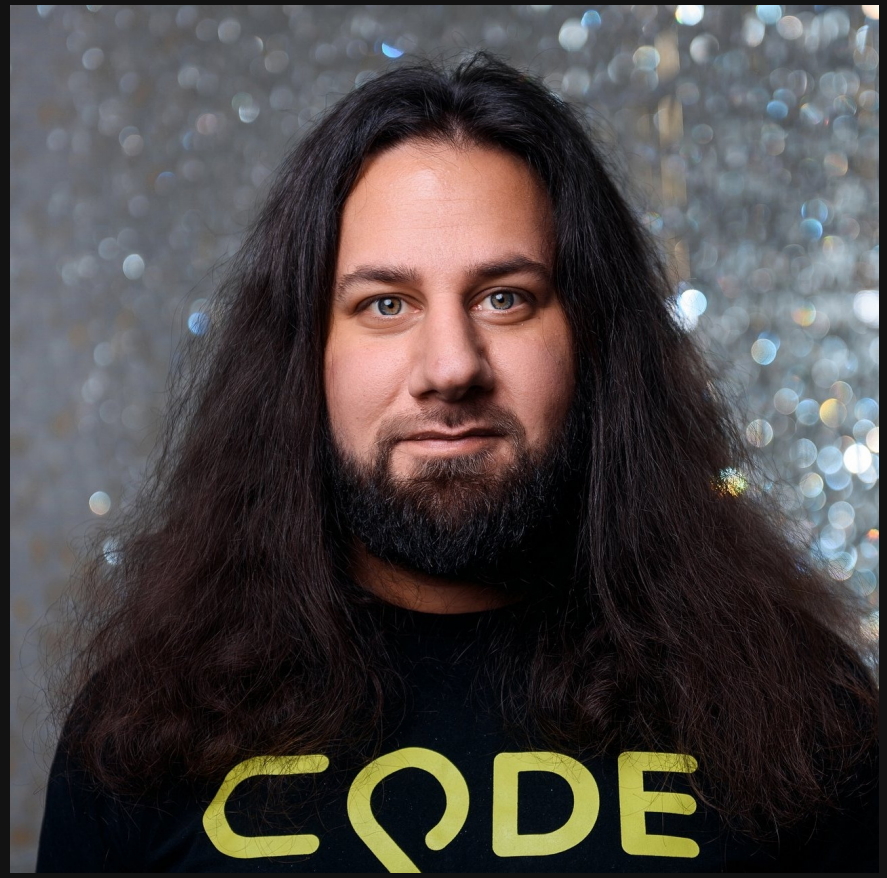
Kotlin Goodies for Testing

Pasha Finkelshteyn, 

`whoami`

- Developer at
- Average enjoyer
- Speaker and streamer (https://twitch.tv/jetbrains or .tv/)
- Data engineer



All programming activities have one common trait

They require testing!

All programming activities have one common trait

They require testing!

In this talk I’m trying to tell about good autotests.
With 

How test in Java usually look?

```
class MyVeryImportantTest {
    @Test
    void feature_should_work_somehow(){
       /*snip*/
       assert
    }
    @Test
    void feature_should_work_somehow2(){
       /*snip*/
       assert
    }
    @Test
    void feature_should_work_somehow3(){
       /*snip*/
       assert
    }
}
```

Many, many times!

How test in Java usually look?

```
class MyVeryImportantTest {
    @Test
    void feature_should_work_somehow(){
       /*snip*/
       assert
    }
    @Test
    void feature_should_work_somehow2(){
       /*snip*/
       assert
    }
    @Test
    void feature_should_work_somehow3(){
       /*snip*/
       assert
    }
}
```

Many, many times!

How test in Java usually look?

```
class MyVeryImportantTest {
    @Test
    void feature_should_work_somehow(){
       /*snip*/
       assert
    }
    @Test
    void feature_should_work_somehow2(){
       /*snip*/
       assert
    }
    @Test
    void feature_should_work_somehow3(){
       /*snip*/
       assert
    }
}
```

Many, many times!

How test in Java usually look?

```
1   class MyVeryImportantTest {
2       @Test
3       void feature_should_work_somehow(){
4          /*snip*/
5          assert
6       }
7       @Test
8       void feature_should_work_somehow2(){
9          /*snip*/
10         assert
11      }
12      @Test
13      void feature_should_work_somehow3(){
14         /*snip*/
15         assert
16      }
17  }
```

Many, many times!

How test in Java usually look?

```
class MyVeryImportantTest {
    @Test
    void feature_should_work_somehow(){
       /*snip*/
       assert
    }
    @Test
    void feature_should_work_somehow2(){
       /*snip*/
       assert
    }
    @Test
    void feature_should_work_somehow3(){
       /*snip*/
       assert
    }
}
```

Many, many times!

How test in Java usually look?

```
1   class MyVeryImportantTest {
2       @Test
3       void feature_should_work_somehow(){
4          /*snip*/
5          assert
6       }
7       @Test
8       void feature_should_work_somehow2(){
9          /*snip*/
10         assert
11      }
12      @Test
13      void feature_should_work_somehow3(){
14         /*snip*/
15         assert
16      }
17  }
```

Many, many times!

How test in Java usually look?

```
class MyVeryImportantTest {
    @Test
    void feature_should_work_somehow(){
       /*snip*/
       assert
    }
    @Test
    void feature_should_work_somehow2(){
       /*snip*/
       assert
    }
    @Test
    void feature_should_work_somehow3(){
       /*snip*/
       assert
    }
}
```

Many, many times!

How test in Java usually look?

```
1   class MyVeryImportantTest {
2       @Test
3       void feature_should_work_somehow(){
4          /*snip*/
5          assert
6       }
7       @Test
8       void feature_should_work_somehow2(){
9          /*snip*/
10         assert
11      }
12      @Test
13      void feature_should_work_somehow3(){
14         /*snip*/
15         assert
16      }
17  }
```

Many, many times!


The Flat World

But we wanna beauty!

```
passwordService {
    shouldHash {
        length { /* snip */ }
        salt { /* snip  */ }
    }
    shouldCheck {
        validPasswords()
        invalidPasswords()
    }
}
```

But we wanna beauty!

```
passwordService {
    shouldHash {
        length { /* snip */ }
        salt { /* snip  */ }
    }
    shouldCheck {
        validPasswords()
        invalidPasswords()
    }
}
```

But we wanna beauty!

```
passwordService {
    shouldHash {
        length { /* snip */ }
        salt { /* snip  */ }
    }
    shouldCheck {
        validPasswords()
        invalidPasswords()
    }
}
```

But we wanna beauty!

```
passwordService {
    shouldHash {
        length { /* snip */ }
        salt { /* snip  */ }
    }
    shouldCheck {
        validPasswords()
        invalidPasswords()
    }
}
```

But we wanna beauty!

```
passwordService {
    shouldHash {
        length { /* snip */ }
        salt { /* snip  */ }
    }
    shouldCheck {
        validPasswords()
        invalidPasswords()
    }
}
```

But we wanna beauty!

```
passwordService {
    shouldHash {
        length { /* snip */ }
        salt { /* snip  */ }
    }
    shouldCheck {
        validPasswords()
        invalidPasswords()
    }
}
```

But we wanna beauty!

```
passwordService {
    shouldHash {
        length { /* snip */ }
        salt { /* snip  */ }
    }
    shouldCheck {
        validPasswords()
        invalidPasswords()
    }
}
```

But we wanna beauty!

```
passwordService {
    shouldHash {
        length { /* snip */ }
        salt { /* snip  */ }
    }
    shouldCheck {
        validPasswords()
        invalidPasswords()
    }
}
```

Parametrized tests? Python

```python
@pytest.mark.parametrize("test_input,expected", [("3+5", 8), ("2+4", 6), ("6*9", 42)])
def test_eval(test_input, expected):
    assert eval(test_input) == expected
```

`test_input,expected`, really?

Parametrized tests? Python

```python
@pytest.mark.parametrize("test_input,expected", [("3+5", 8), ("2+4", 6), ("6*9", 42)])
def test_eval(test_input, expected):
    assert eval(test_input) == expected
```

`test_input,expected`, really?

Parametrized tests? Python

```python
@pytest.mark.parametrize("test_input,expected", [("3+5", 8), ("2+4", 6), ("6*9", 42)])
def test_eval(test_input, expected):
    assert eval(test_input) == expected
```

`test_input,expected`, really?

Parametrized tests? Java

```java
static Stream<Arguments> stringIntAndListProvider() {
    return Stream.of(
        arguments("apple", 1, Arrays.asList("a", "b")),
        arguments("lemon", 2, Arrays.asList("x", "y"))
    );
}
@ParameterizedTest
@MethodSource("stringIntAndListProvider")
void testWithMultiArgMethodSource(String str,
    int num, List<String> list) {/* snip */}
```

`stringIntAndListProvider`, really?

Parametrized tests? Java

```java
static Stream<Arguments> stringIntAndListProvider() {
    return Stream.of(
        arguments("apple", 1, Arrays.asList("a", "b")),
        arguments("lemon", 2, Arrays.asList("x", "y"))
    );
}
@ParameterizedTest
@MethodSource("stringIntAndListProvider")
void testWithMultiArgMethodSource(String str,
    int num, List<String> list) {/* snip */}
```

`stringIntAndListProvider`, really?

Parametrized tests? Java

```
1   static Stream<Arguments> stringIntAndListProvider() {
2       return Stream.of(
3           arguments("apple", 1, Arrays.asList("a", "b")),
4           arguments("lemon", 2, Arrays.asList("x", "y"))
5       );
6   }
7   @ParameterizedTest
8   @MethodSource("stringIntAndListProvider")
9   void testWithMultiArgMethodSource(String str,
10      int num, List<String> list) {/* snip */}
```

`stringIntAndListProvider`, really?

Parametrized tests? Java

```java
static Stream<Arguments> stringIntAndListProvider() {
    return Stream.of(
        arguments("apple", 1, Arrays.asList("a", "b")),
        arguments("lemon", 2, Arrays.asList("x", "y"))
    );
}
@ParameterizedTest
@MethodSource("stringIntAndListProvider")
void testWithMultiArgMethodSource(String str,
    int num, List<String> list) {/* snip */}
```

`stringIntAndListProvider`, really?

Parametrized tests? Kotlin (with kotest)

```kotlin
fun isPythagTriple(a: Int, b: Int, c: Int): Boolean = a * a + b * b == c * c

// snip…

withData(
  Triple(3, 4, 5),
  Triple(6, 8, 10),
  Triple(8, 15, 17),
  Triple(7, 24, 25)
) { (a, b, c) ->
  isPythagTriple(a, b, c) shouldBe true
}
```


What if
I need to test things I don’t know I need to test?

Never write your own hashing (unless…)

```
withData(
    "love",
    "secret",
    "god", // Who will remember the reference?
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Never write your own hashing (unless…)

```
withData(
    "love",
    "secret",
    "god", // Who will remember the reference?
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Never write your own hashing (unless…)

```
withData(
    "love",
    "secret",
    "god", // Who will remember the reference?
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Never write your own hashing (unless…)

```
withData(
    "love",
    "secret",
    "god", // Who will remember the reference?
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Never write your own hashing (unless…)

```
withData(
    "love",
    "secret",
    "god", // Who will remember the reference?
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Launching…

Never write your own hashing (unless…)

```
withData(
    "love",
    "secret",
    "god", // Who will remember the reference?
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Launching…

1. ✔ love
2. ✔ secret
3. ✔ god

Never write your own hashing (unless…)

```
withData(
    "love",
    "secret",
    "god", // Who will remember the reference?
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Launching…

1. ✔ love
2. ✔ secret
3. ✔ god

Time to investigate!

Property tests!

To test things we can not even *imagine*

```
checkAll(
    Arb.string(
        codepoints = Arb.of(
            (Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
        ),
        range = 0..52
    )
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Property tests!

To test things we can not even *imagine*

```
checkAll(
    Arb.string(
        codepoints = Arb.of(
            (Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
        ),
        range = 0..52
    )
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Property tests!

To test things we can not even *imagine*

```
checkAll(
    Arb.string(
        codepoints = Arb.of(
            (Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
        ),
        range = 0..52
    )
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Property tests!

To test things we can not even *imagine*

```
checkAll(
    Arb.string(
        codepoints = Arb.of(
            (Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
        ),
        range = 0..52
    )
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Property tests!

To test things we can not even *imagine*

```
checkAll(
    Arb.string(
        codepoints = Arb.of(
            (Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
        ),
        range = 0..52
    )
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Property tests!

To test things we can not even *imagine*

```
checkAll(
    Arb.string(
        codepoints = Arb.of(
            (Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
        ),
        range = 0..52
    )
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Property tests!

To test things we can not even *imagine*

```
checkAll(
    Arb.string(
        codepoints = Arb.of(
            (Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
        ),
        range = 0..52
    )
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Property tests!

To test things we can not even *imagine*

```
checkAll(
    Arb.string(
        codepoints = Arb.of(
            (Char.MIN_VALUE.code..Char.MAX_VALUE.code).map(::Codepoint)
        ),
        range = 0..52
    )
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

Error!

```
Property test failed for inputs

0) "꿟ル渼啣ຣᓄ驓拒③ஹ꽌⍰⍰滤砑琶10月뉥寵巘뗑筞醉°F२쫡Ấ鼏켓鹗맵檍ở儋⋉綦講뾧仈ी豳鄙ῇ悤川陜盜"

Caused by io.kotest.assertions.AssertionFailedError: expected:<"90b23a970ed1b799850a24140ae8ad39"> but was:<"5a430f76b3d544a1
```

Wow, it was a complex sample!

But there is more…

Error!

```
Property test failed for inputs

0) "쨇ภ渊啣ຮᴗ驔拒③ൠ괖⍰⍰熄砑琶10月뇷寵嚱땡筞醉°Fʢ쫡Ấ鼏켓鶚맥櫶ở僖ⅹ綦講喫仦ᡢ豳鄙ŋ̤悤川陜盗"

Caused by io.kotest.assertions.AssertionFailedError: expected:<"90b23a970ed1b799850a24140ae8ad39"> but was:<"5a430f76b3d544a1
```

Wow, it was a complex sample!

But there is more…

Error!

```
Property test failed for inputs

0) "꿟ᥰ湵啣ຮں騿拒③ဤ끞□□㴱砑琶㋉늙寵嚱뜷筞醉°Fર쫗Ẳ鼏켯鶚맵橞ở僑⋉綦講嬰仦ᡳ豳鄙ῃ恴川陜盜"

Caused by io.kotest.assertions.AssertionFailedError: expected:<"90b23a970ed1b799850a24140ae8ad39"> but was:<"5a430f76b3d544a1
```

Wow, it was a complex sample!

But there is more…

Error!

```
Property test failed for inputs

0) "꽭ᆙ溡啣ຣᓑ騿挋③ஹ꾡⍰⍰滬砑琶㏙뇛寵巇떍箣醉°Fʢ쫡Ẩ鼏켗鶚맴槵ở儵ϰ綦講쀚伱ᡣ豂鄙ŋ̣惪川陝盗"

Caused by io.kotest.assertions.AssertionFailedError: expected:<"90b23a970ed1b799850a24140ae8ad39"> but was:<"5a430f76b3d544a1
```

Wow, it was a complex sample!

But there is more…

Error!

```
Attempting to shrink arg "꿿ル湺啣ຮᑌ騿拒③ဤ괖□□潩砑琶㋉뇗竉嚱떪筴醉°Fʾ쫔Ẩ鼏켔鶚맥檍ở儋⋉綦講㜳仦ី豳鄙ŋ̇怣川陝盗"
Shrink #1: "꿿ル湺啣ຮᑌ騿拒③ဤ괖□□潩砑琶㋉뇗竉嚱떪筴醉°F" fail
Shrink #2: "꿿ル湺啣ຮᑌ騿拒③ဤ괖✷" fail
Shrink #3: "꿿ル湺啣ຮᑌ" fail
Shrink #4: "꿿ル湺" fail
Shrink #5: "꿿ル" fail
Shrink #6: "꿿" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "꿿"
```

And there is more!

Error!

```
Attempting to shrink arg "꿟ル渕啣ຮᴗ騿挋③ဤ곺⍰⍰㵍砑琶㋉놡竉嚱떱筞醉°Fʢ쫙Ẩ鼏켗鶚맥檍ở僐⋉綦講뺕仦ꦿ豳鄙ŋ̣悳川陜盗"
Shrink #1: "꿟ル渕啣ຮᴗ騿挋③ဤ곺⍰⍰㵍砑琶㋉놡竉嚱떱筞醉°F" fail
Shrink #2: "꿟ル渕啣ຮᴗ騿挋③ဤ곺✴" fail
Shrink #3: "꿟ル渕啣ຮᴗ" fail
Shrink #4: "꿟ル渕" fail
Shrink #5: "꿟ル" fail
Shrink #6: "꿟" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "꿟"
```

And there is more!

Error!

```
Attempting to shrink arg "꾧ル渫啣ᥪᴗ騿拒③ஹ끖⍰⍰滮砑琶㋉뇛寵嚱뗂筴醉°Fʢ쫝Ȃ鼏켓鶚맵檍ở僐⋉綦講嬰仦ৰ豳鄙ŋ̣悥川陝盜"
Shrink #1: "꾧ル渫啣ᥪᴗ騿拒③ஹ끖⍰⍰滮砑琶㋉뇛寵嚱뗂筴醉°F" fail
Shrink #2: "꾧ル渫啣ᥪᴗ騿拒③ஹ끖✷" fail
Shrink #3: "꾧ル渫啣ᥪᴗ" fail
Shrink #4: "꾧ル渫" fail
Shrink #5: "꾧ル" fail
Shrink #6: "꾧" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "꾧"
```

And there is more!

Error!

```
Attempting to shrink arg "꽪ㄦ浂啣ຣں騿拒③ஜ괖⍰⍰淴砑琶㋉놣竉嚱뜹箂醉°Fʢ쫚Ẩ鼏켗鶚맵橞ở僐ϰ綦講嬰仦ᦓ黺鄙ŋ̈恴川陝盗"
Shrink #1: "꽪ㄦ浂啣ຣں騿拒③ஜ괖⍰⍰淴砑琶㋉놣竉嚱뜹箂醉°F" fail
Shrink #2: "꽪ㄦ浂啣ຣں騿拒③ஜ괖✷" fail
Shrink #3: "꽪ㄦ浂啣ຣں" fail
Shrink #4: "꽪ㄦ浂" fail
Shrink #5: "꽪ㄦ" fail
Shrink #6: "꽪" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "꽪"
```

And there is more!

Error!

```
Attempting to shrink arg "꽪ㄦ滕啣ຮت騲拒③ஜ괖□□瀄砑琶㋉넗寵嚱땷筴醉°Fʑ쫚Ẩ鼏켲鶚맵槵ở僐ϰ綦講奬仦ꫩ黺鄙ŋ̈息川陝盗"
Shrink #1: "꽪ㄦ滕啣ຮت騲拒③ஜ괖□□瀄砑琶㋉넗寵嚱땷筴醉°F" fail
Shrink #2: "꽪ㄦ滕啣ຮت騲拒③ஜ괖✷" fail
Shrink #3: "꽪ㄦ滕啣ຮت" fail
Shrink #4: "꽪ㄦ滕" fail
Shrink #5: "꽪ㄦ" fail
Shrink #6: "꽪" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "꽪"
```

And there is more!

Error!

```
Attempting to shrink arg "꾧ﾙ湵啣ຣᵕ騿拒③ည곾⍰⍰㴧砑琶㏩늚竉嚱땧筞醉°Fʔ짝Ẩ鼏켓鶚맥檍ở儋ꭍ綦講喫仦ৗ豳鄙ῃ悳川陜盗"
Shrink #1: "꾧ﾙ湵啣ຣᵕ騿拒③ည곾⍰⍰㴧砑琶㏩늚竉嚱땧筞醉°F" fail
Shrink #2: "꾧ﾙ湵啣ຣᵕ騿拒③ည곾✷" fail
Shrink #3: "꾧ﾙ湵啣ຣᵕ" fail
Shrink #4: "꾧ﾙ湵" fail
Shrink #5: "꾧ﾙ" fail
Shrink #6: "꾧" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "꾧"
```

And there is more!

Error!

```
Attempting to shrink arg "꽳ル湙啣ຣᥙ騿拒③ꩬ괖??㴧砑琶㋉뵗寵嚱떯箂醉℉ʢ쫅Ẩ鼏켗鶚맥槵ở僐ϰ綦講單仦ᨪ豳鄙ŋ̣息川陝盗"
Shrink #1: "꽳ル湙啣ຣᥙ騿拒③ꩬ괖??㴧砑琶㋉뵗寵嚱떯箂醉℉" fail
Shrink #2: "꽳ル湙啣ຣᥙ騿拒③ꩬ괖✷" fail
Shrink #3: "꽳ル湙啣ຣᥙ" fail
Shrink #4: "꽳ル湙" fail
Shrink #5: "꽳ル" fail
Shrink #6: "꽳" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "꽳"
```

And there is more!

Error!

```
Attempting to shrink arg "꿿ᥰ渕啣ຣᓑ騿拒③ꦗ괖⍰⍰滺砑琶㏩뇷寵曦땽筴醉°F꒶쬒Ẩ冪켔鴞맥檍ở僖ϰ綦講㛙仦ᯤ豳鄙ŋ̣悳川陜盔"
Shrink #1: "꿿ᥰ渕啣ຣᓑ騿拒③ꦗ괖⍰⍰滺砑琶㏩뇷寵曦땽筴醉°F" fail
Shrink #2: "꿿ᥰ渕啣ຣᓑ騿拒③ꦗ괖✷" fail
Shrink #3: "꿿ᥰ渕啣ຣᓑ" fail
Shrink #4: "꿿ᥰ渕" fail
Shrink #5: "꿿ᥰ" fail
Shrink #6: "꿿" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "꿿"
```

And there is more!

Error!

```
Attempting to shrink arg "꾧ル㴵啣ຣں騿拒③ꦝ괖⍰⍰㴧砑琶㏪뇛竉嚱띱筞醉°Fʢ짴Ẩ鼏켗鶚맥槵ở僖ϰ綦講嚶仦ั豳鄙ῃ悳川陜盗"
Shrink #1: "꾧ル㴵啣ຣں騿拒③ꦝ괖⍰⍰㴧砑琶㏪뇛竉嚱띱筞醉°F" fail
Shrink #2: "꾧ル㴵啣ຣں騿拒③ꦝ괖✷" fail
Shrink #3: "꾧ル㴵啣ຣں" fail
Shrink #4: "꾧ル㴵" fail
Shrink #5: "꾧ル" fail
Shrink #6: "꾧" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "꾧"
```

And there is more!

Error!

```
Attempting to shrink arg "꽹ル湵啣ຣں騿挋③ဤ괖⍰⍰潩砑琶㋉늻竉嚱떕筞醉°FʢÂ쫚鼏켗鶚맵檍ở僐ᶍ綦講噢仦ᦂ豳鄙ŋ̉悥川陝盗"
Shrink #1: "꽹ル湵啣ຣں騿挋③ဤ괖⍰⍰潩砑琶㋉늻竉嚱떕筞醉°F" fail
Shrink #2: "꽹ル湵啣ຣں騿挋③ဤ괖✷" fail
Shrink #3: "꽹ル湵啣ຣں" fail
Shrink #4: "꽹ル湵" fail
Shrink #5: "꽹ル" fail
Shrink #6: "꽹" fail
Shrink #7: <empty string> pass
Shrink result (after 7 shrinks) => "꽹"
```

And there is more!

Error!

```
Property failed after 1 attempts

  Arg 0: "괭" (shrunk from 괭ㅿ溎啣ຣ౿騿搄③ꩡ괖⍰⍰滬砑琶㏠뇷寵巘땷筞醉°Fʔ쬒Ẩ鼏켗鶚맵檍ở僪⋉綦講뽖仦ꦾ黺鄙ῃ悳川陜盀)

Repeat this test by using seed 4919313423746630853
```

Error!

```
Property failed after 1 attempts

  Arg 0: "꽲" (shrunk from 꽲ル滌啣ຮں騿拒③ᨍᨏ괅⍰⍰滬砑琶⑽뇷寵巇땫筞醉°Fʅ쫚Ȁ鼏켓鶚맵檍ở僑⋉綦講螧仦ន豩鄙ῄ悤川陜盔)

Repeat this test by using seed 4919313423746630853
```

Error!

```
Property failed after 1 attempts

  Arg 0: "꾕" (shrunk from 꾕ㅆ滐啣ȣ￼驜拒③ဤ괖￼￼滤砑琶㏨뇘寵巘떪筞醉°F⥿쫙Ẩ鼏켔鶚맥槵ở僐⋉綦講뾏仈ǹ粉鄙ῃ悥川陝盗)

Repeat this test by using seed 4919313423746630853
```

Error!

```
Property failed after 1 attempts

  Arg 0: "꽪" (shrunk from 꽪ㅅ濒啣ꜱ◡驊挋③ᨐ괖⊡⊡溳砑琶10朋늷竈囌땒箂醉°Fᵡ꽑Ấ鼏켓鶚맼橞ð鵽ᵪ綦講뿆仆ᵮ黝鄙ῆ惹川陝盜)

Repeat this test by using seed 49193134237466308 53
```

Repeat? Yes!

```
checkAll(
    PropTestConfig(seed = 4919313423746630853),
    Arb.string(
        codepoints = Arb.of(
            ('a'.code..Char.MAX_HIGH_SURROGATE.code)
                .map(::Codepoint)
        ),
    )
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

- reproduce the test **exactly** like in the failed instance
- debug it
- fix it
- PROFIT!!

Repeat? Yes!

```
checkAll(
    PropTestConfig(seed = 4919313423746630853),
    Arb.string(
        codepoints = Arb.of(
            ('a'.code..Char.MAX_HIGH_SURROGATE.code)
                .map(::Codepoint)
        ),
    )
) { input ->
    MD5.compute(input) shouldBe libMd5(input)
}
```

- reproduce the test **exactly** like in the failed instance
- debug it
- fix it
- PROFIT!!