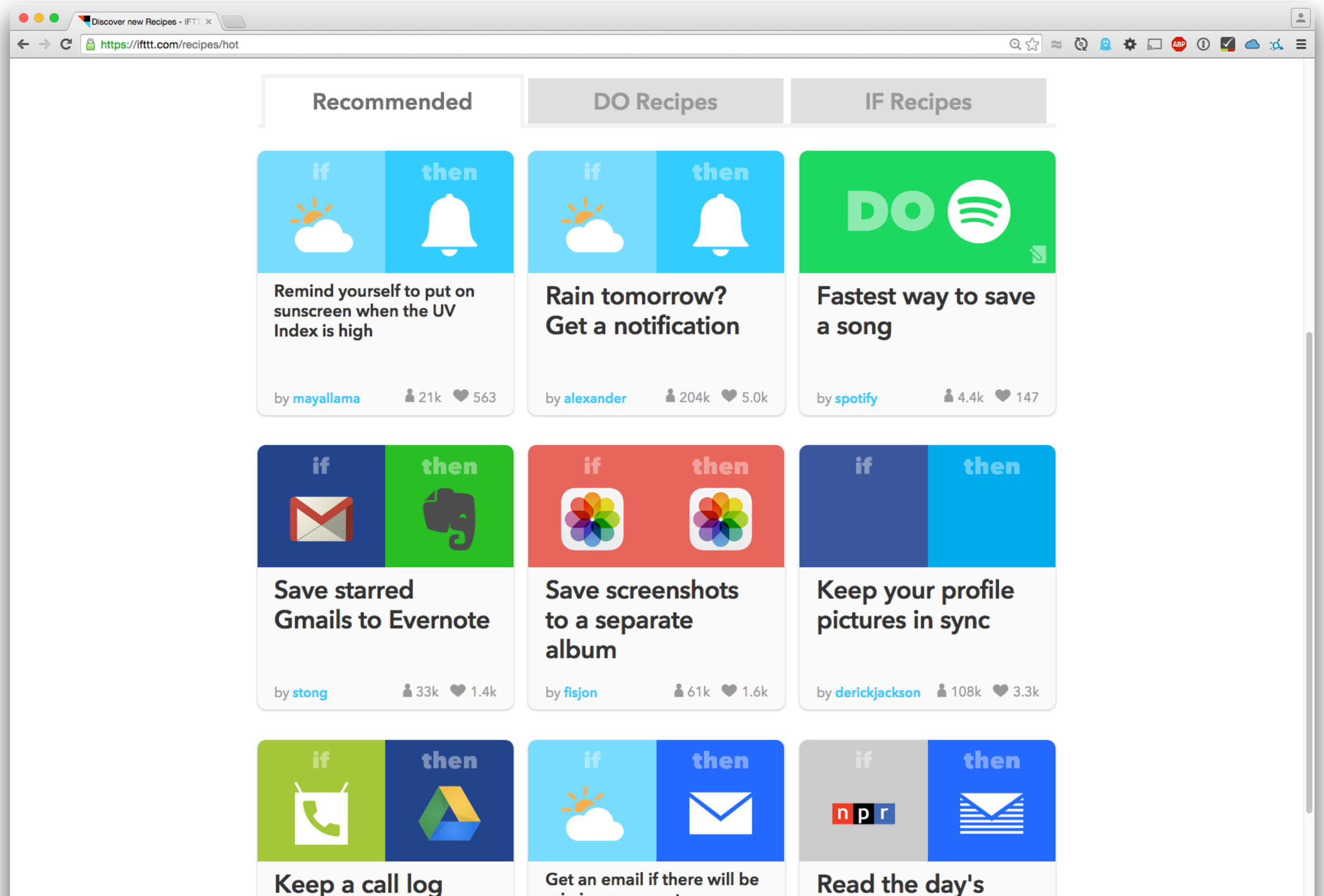
Internal DSLs: Control-Flow

Is this a DSL?

If This Then That (IFTT)



Simple techniques for adding fluency

Most general-purpose languages support these features.

names

including Unicode

```
sin(0)
```

ASK: If the DSL supports Unicode, how will the user write programs?

whitespace

```
computer();
  processor();
    cores(2);
  disk();
    size(150);
```

function composition

```
computer(
  processor(
    cores(2)
  ),
  disk(
    size(150)
  )
);
```

method chaining

```
computer()
  .processor()
    .cores(2)
  .disk()
    .size(150)
  .end();
```

Techniques for hiding the host language

These features tend to be language-specific. Some languages support this ability more than others.

(re-)defining operators	`set1 ∪ set2`
	`set1 + set2`
	Different host languages gives us different amounts of control over precedence and associativity.

infix operators	`set1 union set2` `salaries map giveRaise`
pre- and postfix operators	`~1` `i++`
literal extension	`3 little pigs`

closures
curried functions, call-by-name,…
	`test("An empty Set should have size 0") {`
	`  assert(Set.empty.size == 0)`
	`}`
	Useful for defining new **control-flow structures**

Implicit conversions

See *Scala for the Impatient*, Chapter 21.4

The compiler looks for an implicit conversion when:

- the expected type differs from the inferred type
- an object does not contain an expected attribute

The compiler finds an implicit conversion when:

- a conversion is declared as `implicit`
- a conversion is in scope and is named with a single identifier
- a conversion is defined in the current class's *companion object*

The compiler does not look for an implicit conversion when:

- the code compiles without one
- the compiler has already performed one (for a given expression)
- it finds multiple conversions (i.e., conversion is ambiguous)

```
import scala.language.implicitConversions

:implicits [-v]

-Xprint:typer
```

Implementing new control-flow structures

```
var i = 0
while (i < 10) {
    if ( (i % 2) == 0 )
        println(i)
    i += 1
}
```

↦

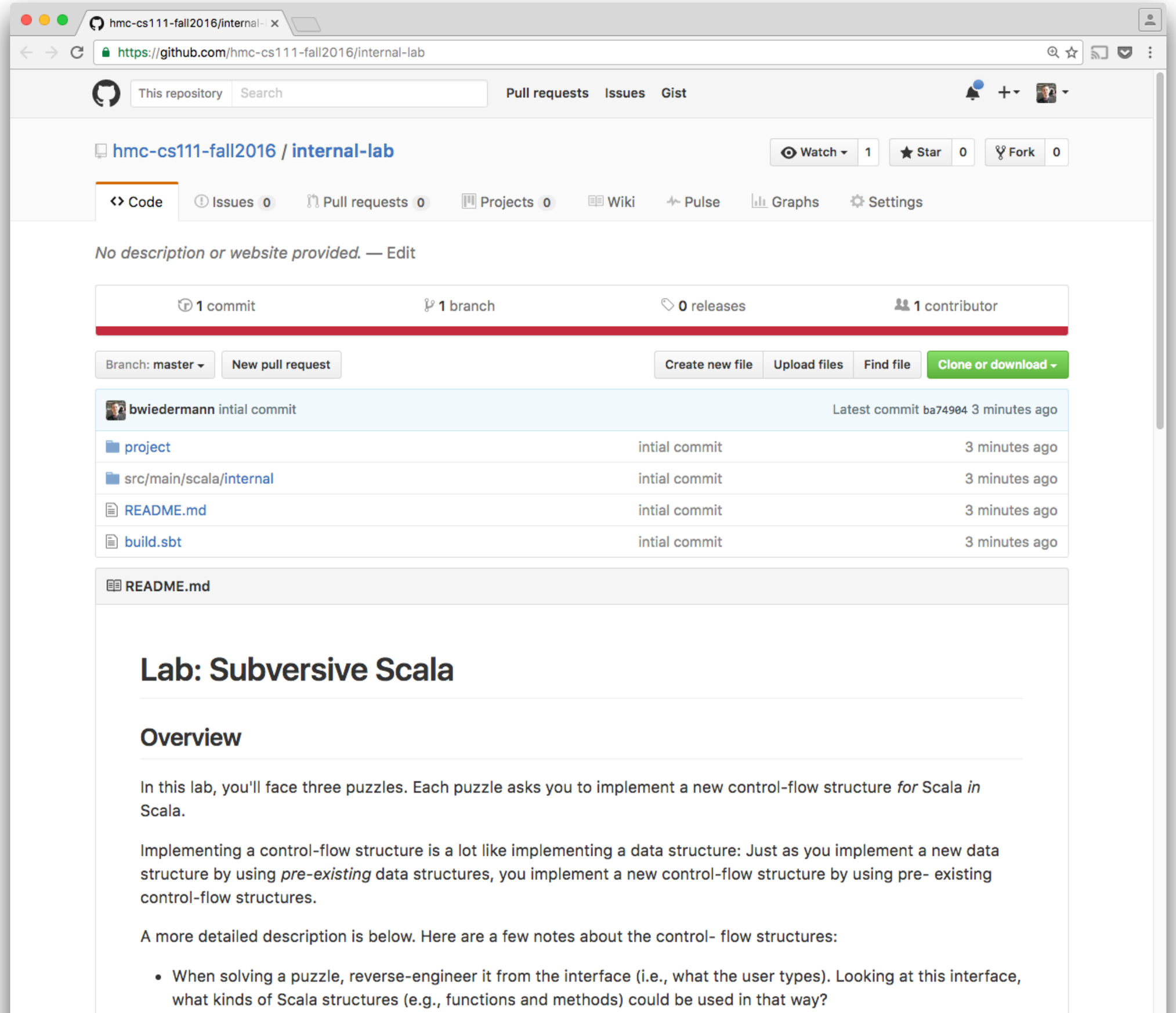
```
var i = 0
loop_until (i > 9) {
    if ( (i % 2) == 0 )
        println(i)
    i += 1
}
```

The Scala-y way to do this is:

```
(0 until 10) filter(_ % 2 == 0) map println
```

but we're feeling subversive today.

github.com/hmc-cs111-fall2016/internal-lab



by 1:55

read overview & set up

by 2:15

implement `loop_until`

by 2:20

implement `repeat_until`

If you thought `goto` was considered harmful…

```
10 COMEFROM 40
20 INPUT "WHAT IS YOUR NAME? "; A$
30 PRINT "HELLO, "; A$
40 REM
```

comes from en.wikipedia.org/wiki/COMEFROM