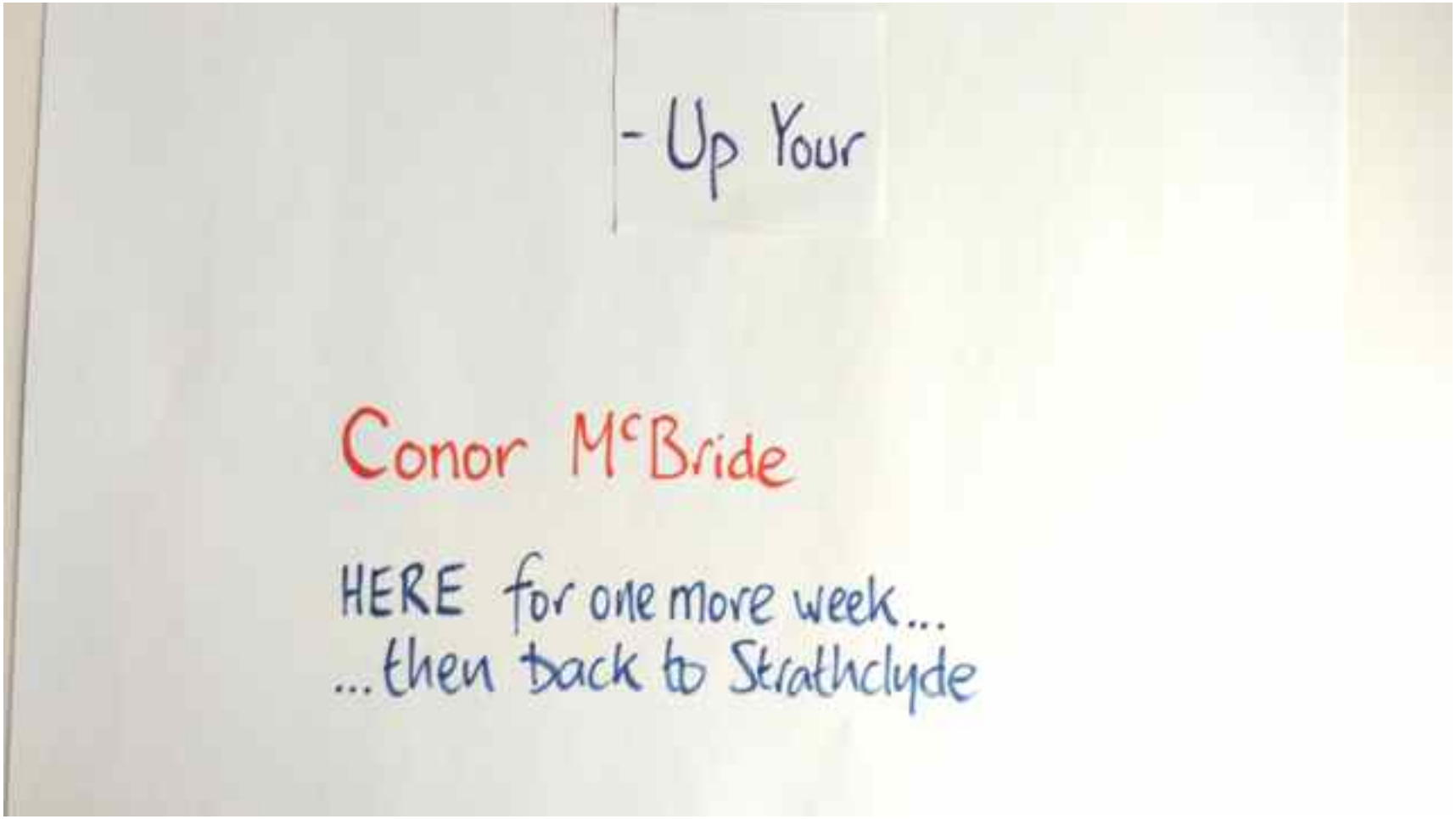
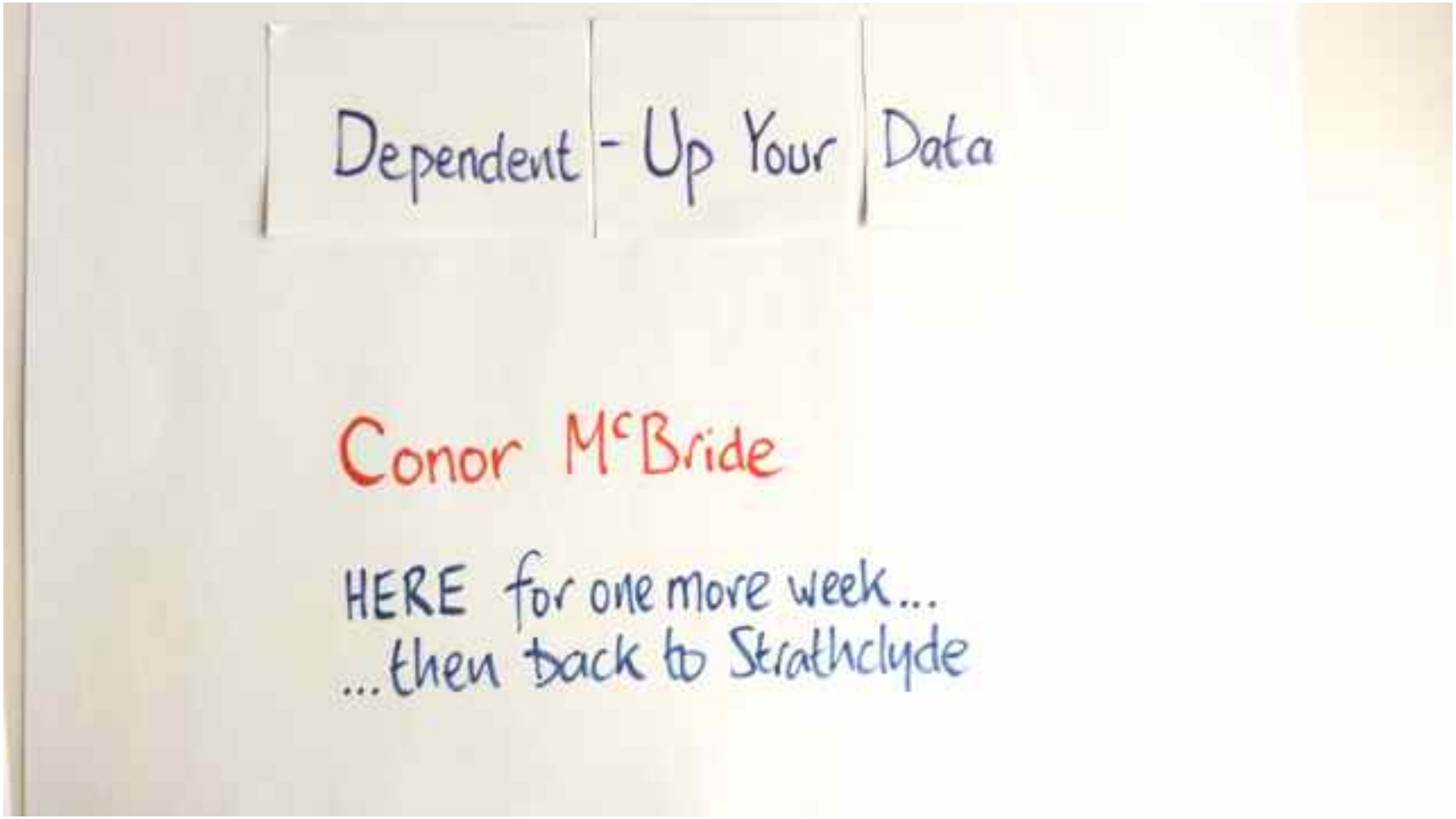
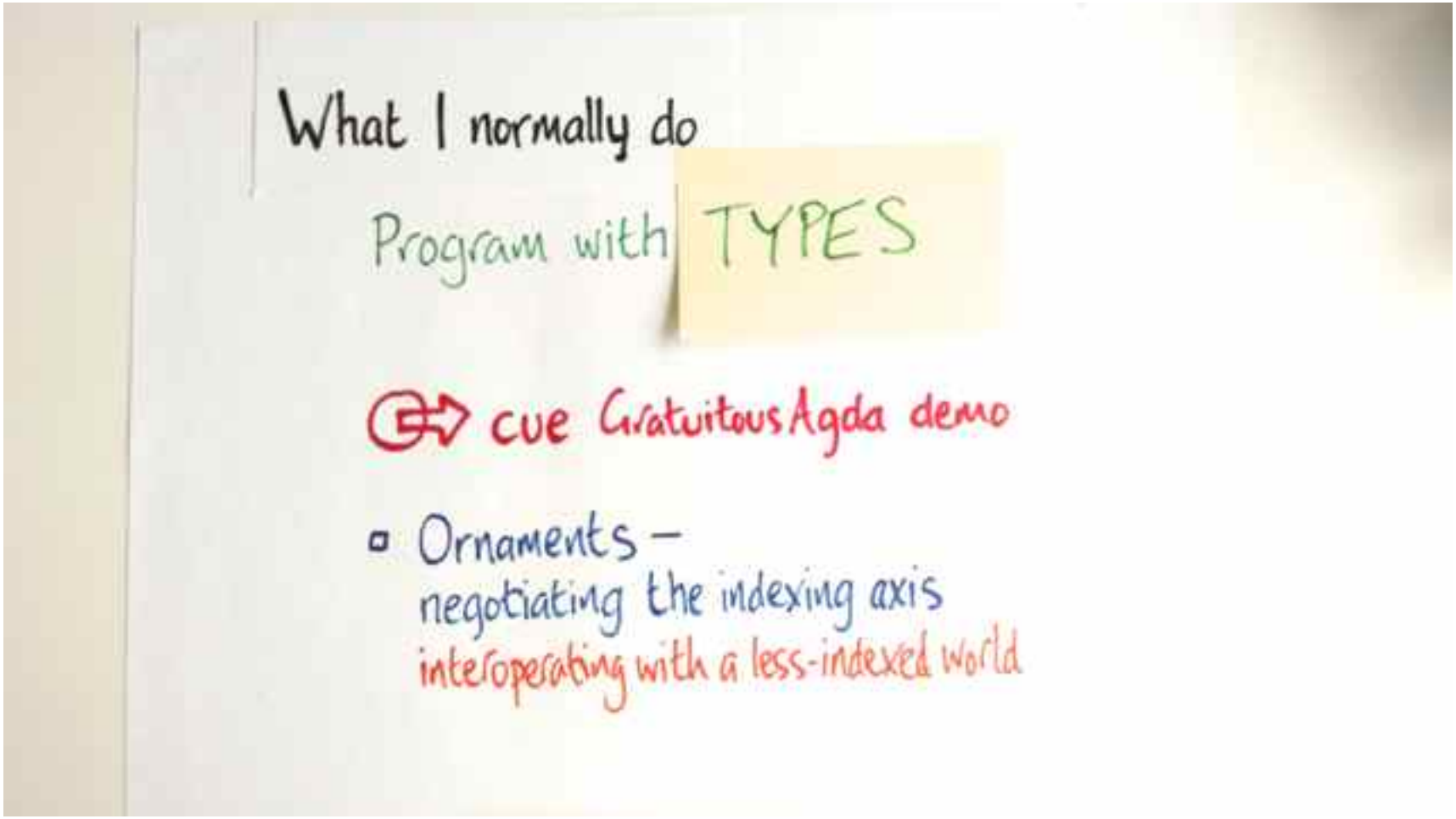
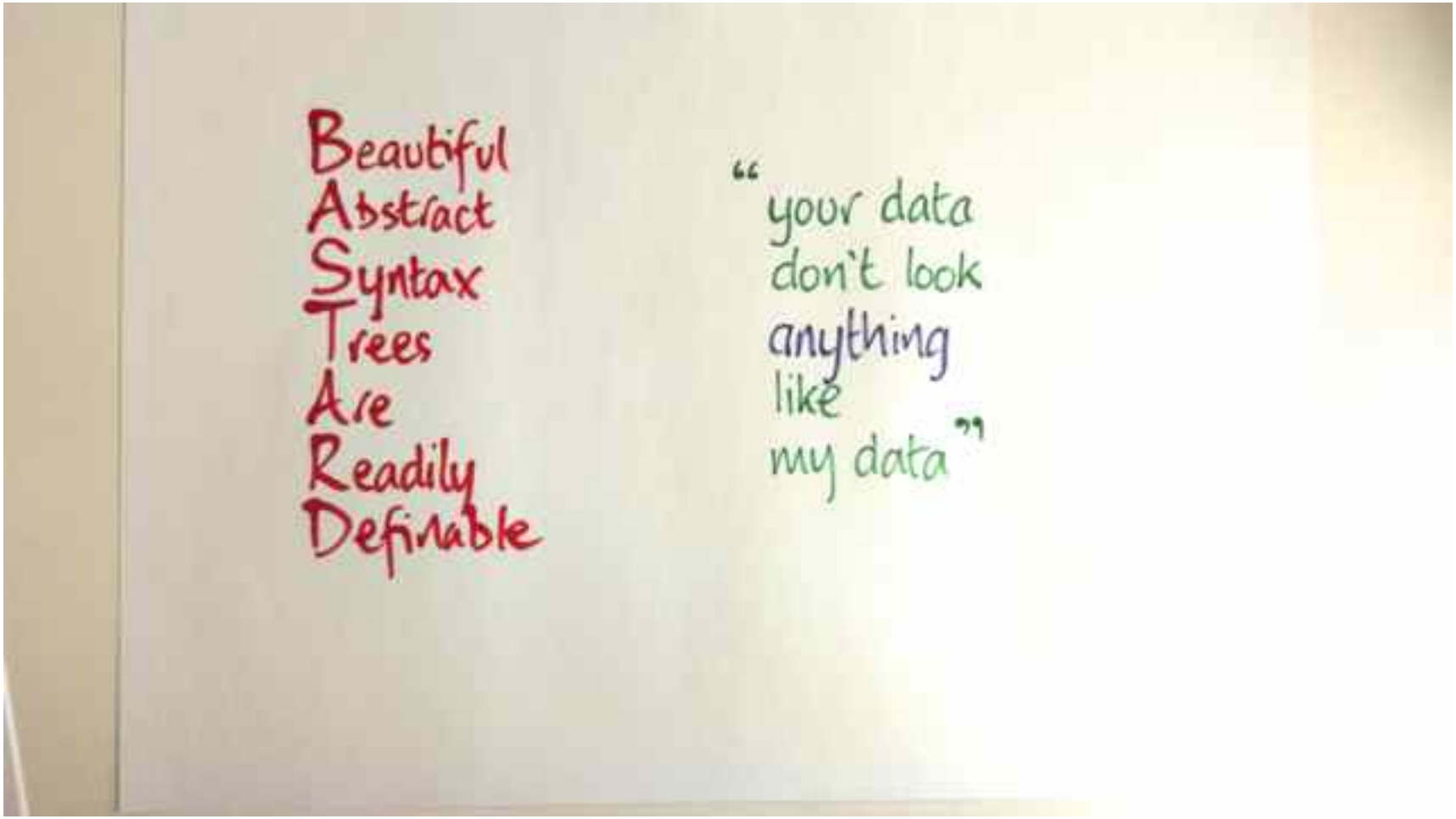
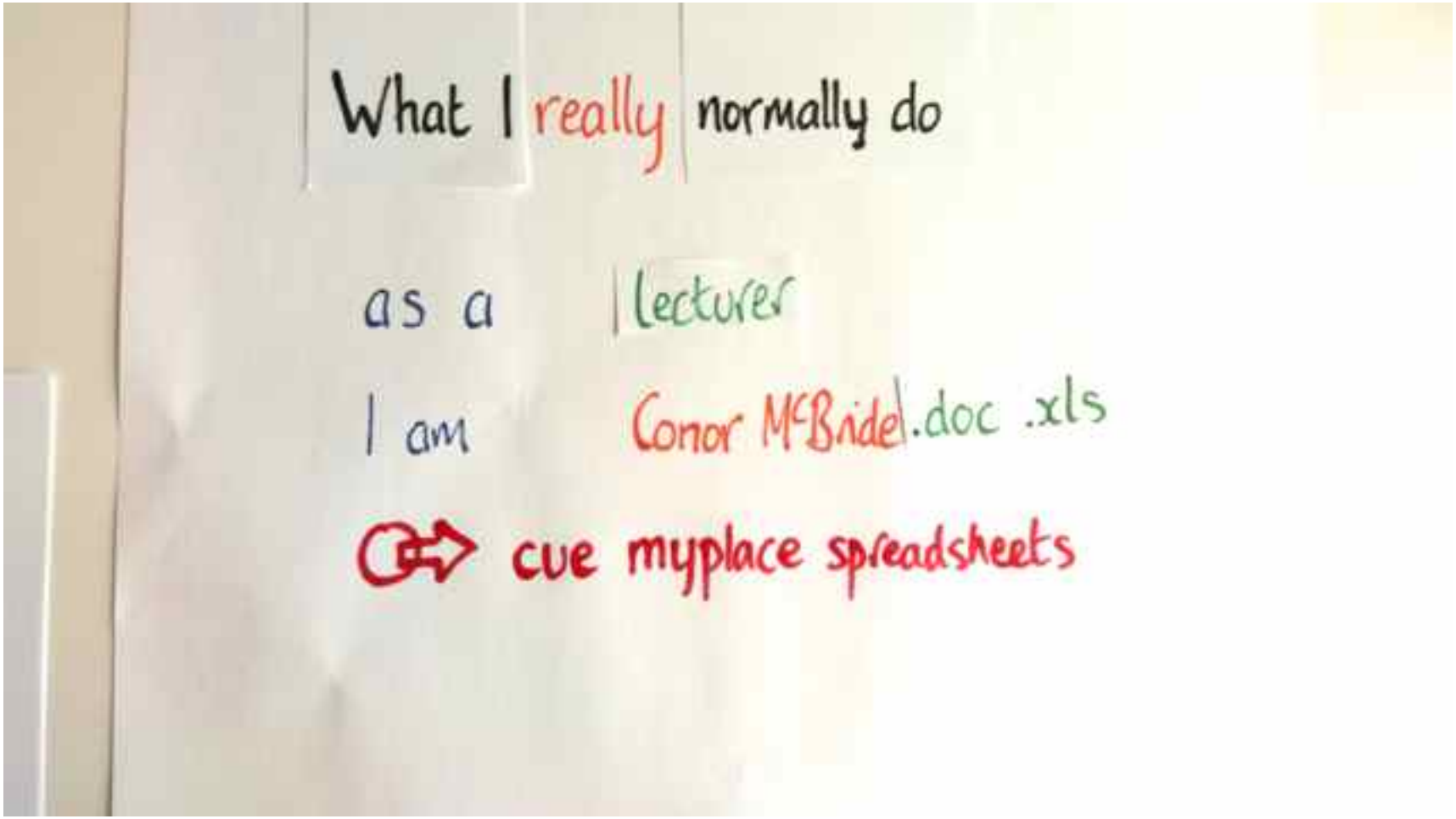

- Up Your
Conor McBride
HERE for one more week...
...then back to Strathclyde

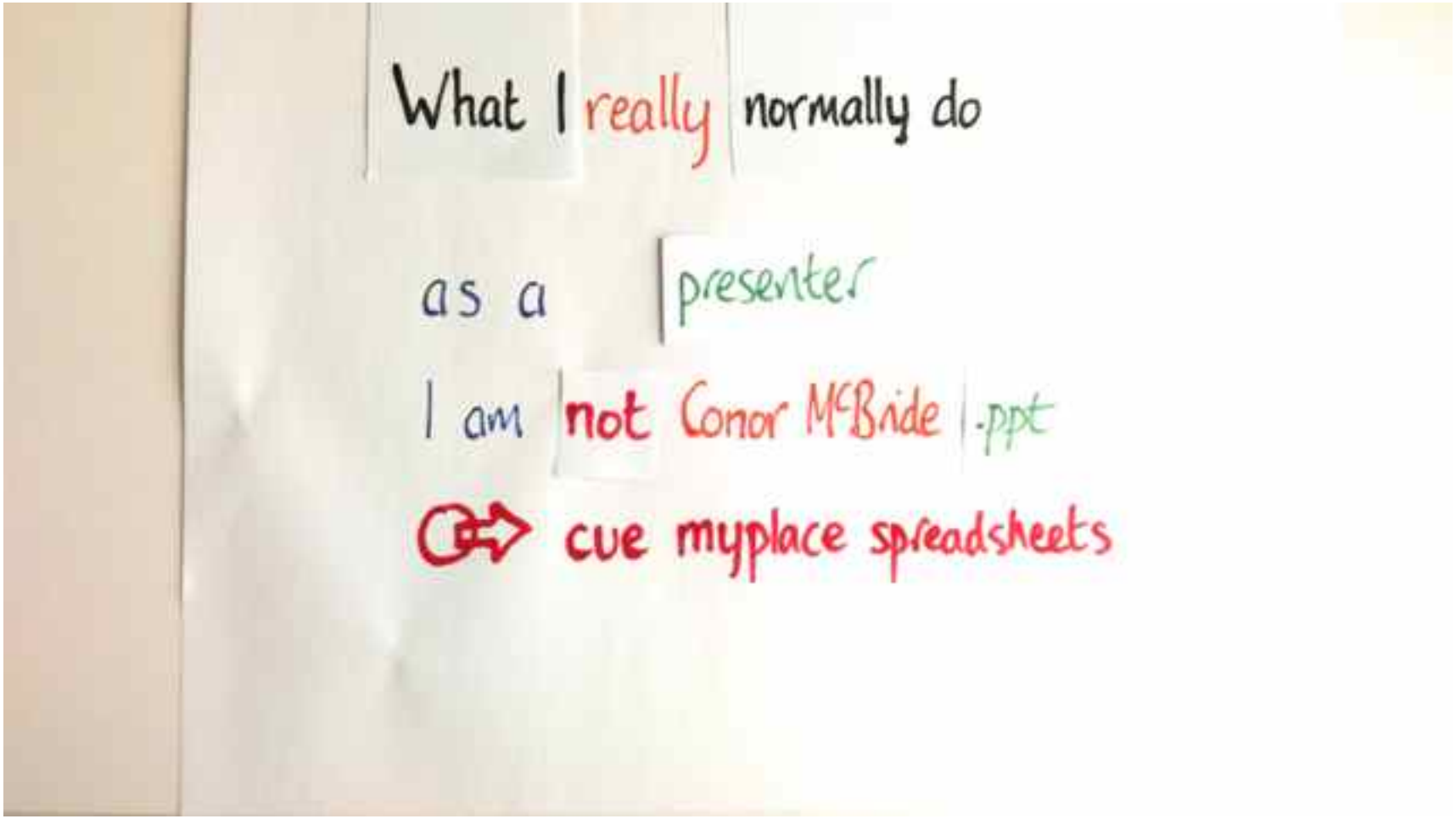

Dependent - Up Your Data
Conor McBride
HERE for one more week...
...then back to Strathclyde


Data - Up Your Dependent
TYPES
Conor McBride
HERE for one more week...
...then back to Strathclyde


What I normally do
Program with TYPES
cue Gratuitous Agda demo
Ornaments –
negotiating the indexing axis
interoperating with a less-indexed world


Beautiful
Abstract
Syntax
Trees
Are
Readily
Definable
"your data
don't look
anything
like
my data"


What I really normally do
as a lecturer
I am Conor McBride .doc .xls
cue myplace spreadsheets


What I really normally do
as a presenter
I am not Conor McBride .ppt
cue myplace spreadsheets

So, after helping to smuggle

dependent TYPES

into Haskell, the obvious

next place to go is

So, after helping to smuggle

dependent TYPES

into Haskell, the obvious

next place to go is Excel.

So, after helping to smuggle

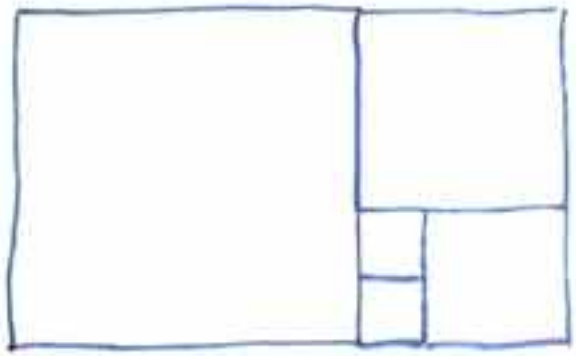
dependent TYPES

into Haskell, the obvious

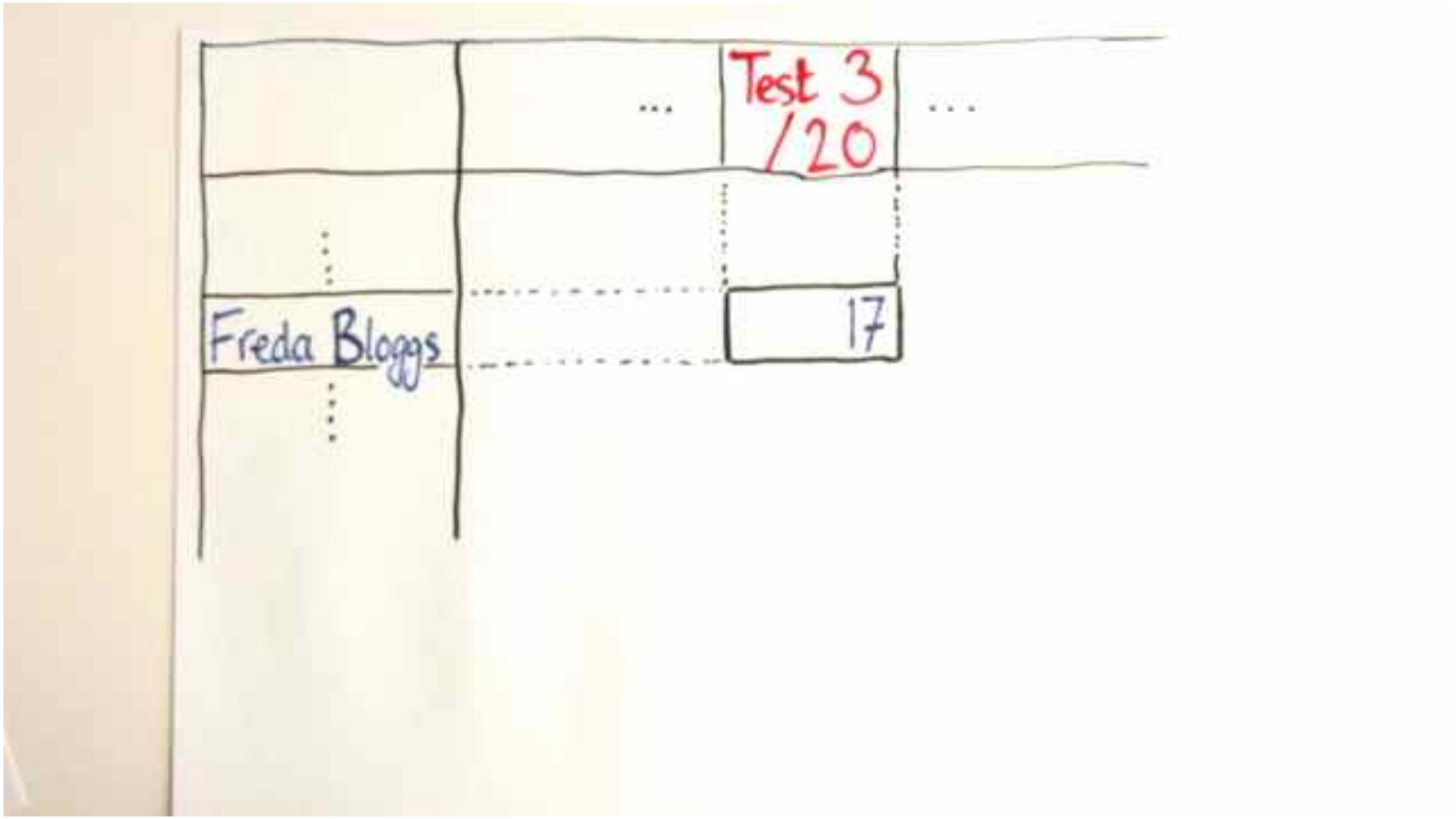
next place to go is Excel.

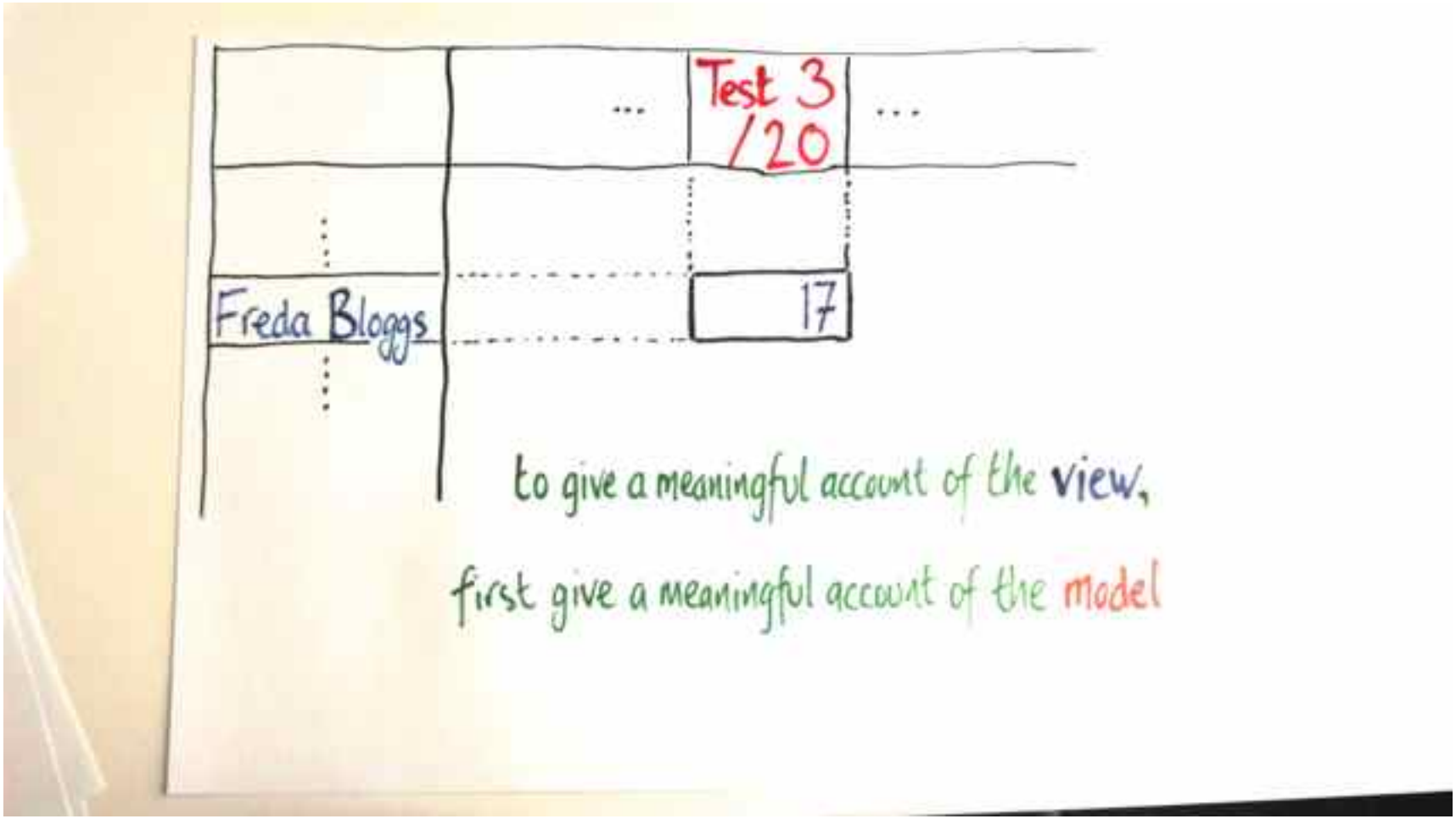
reads lots of papers by Martin Erwig

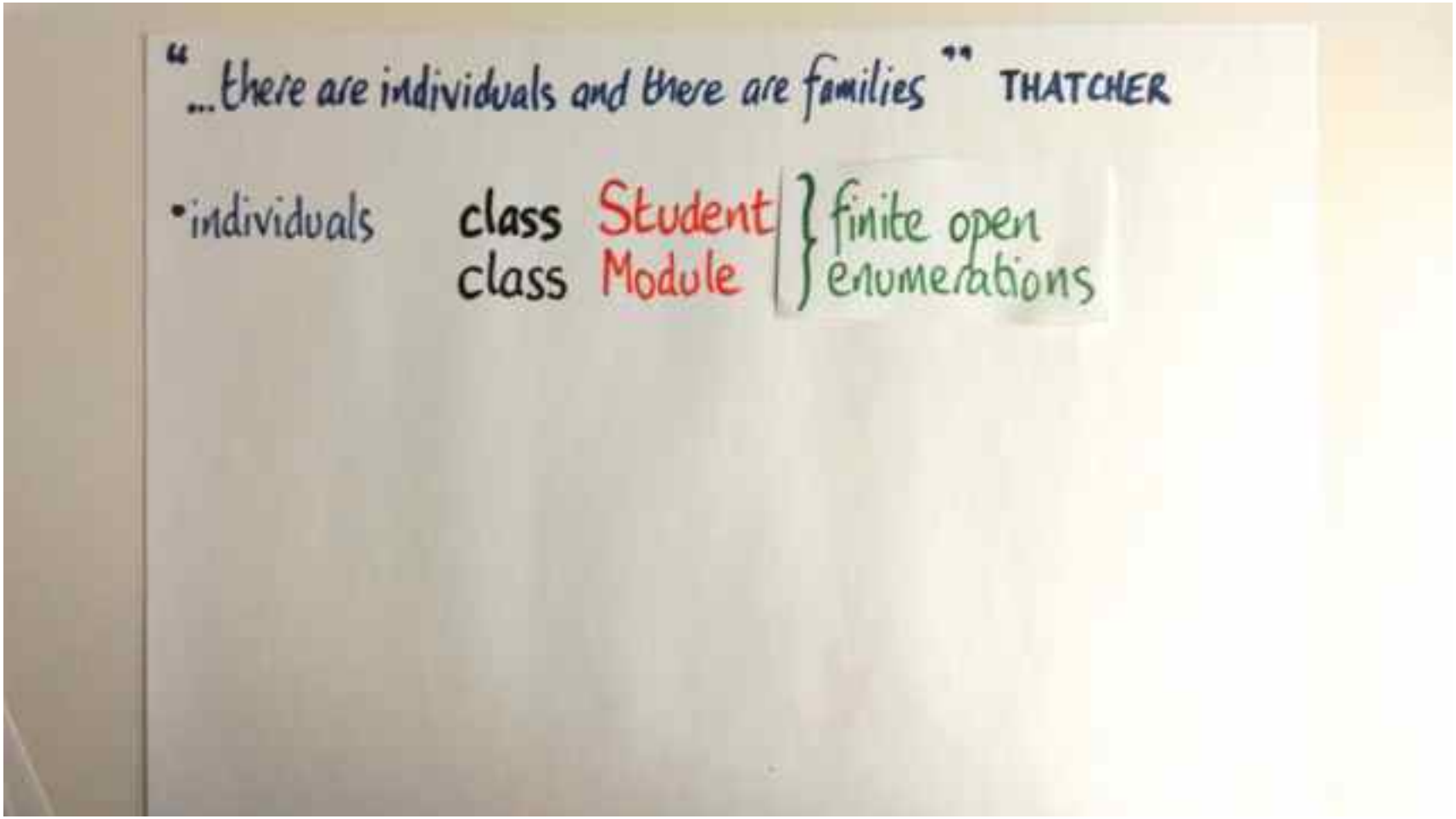
in "Hasochism...", Lindley and I started
The Society for Chopping Up Boxes

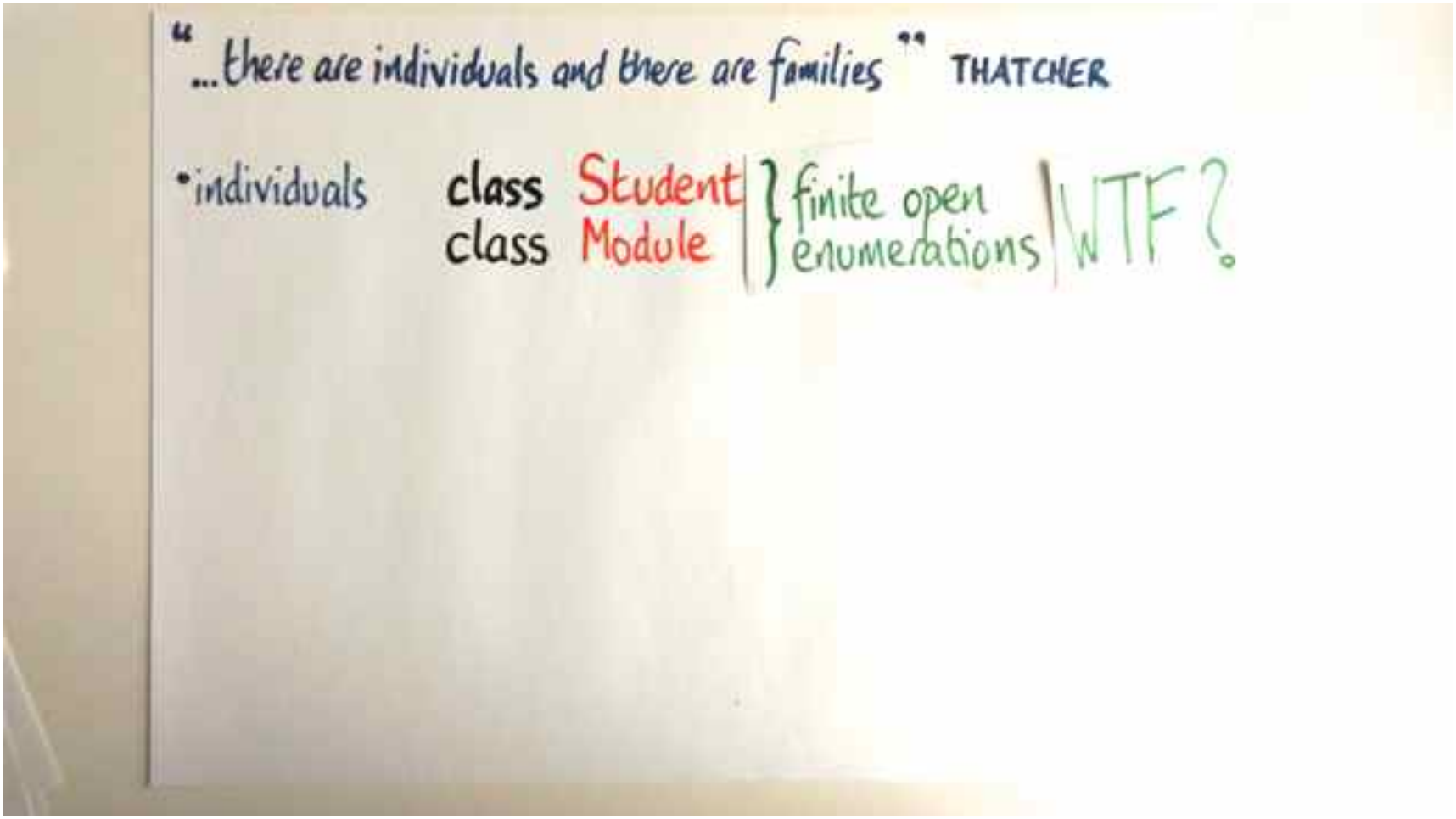


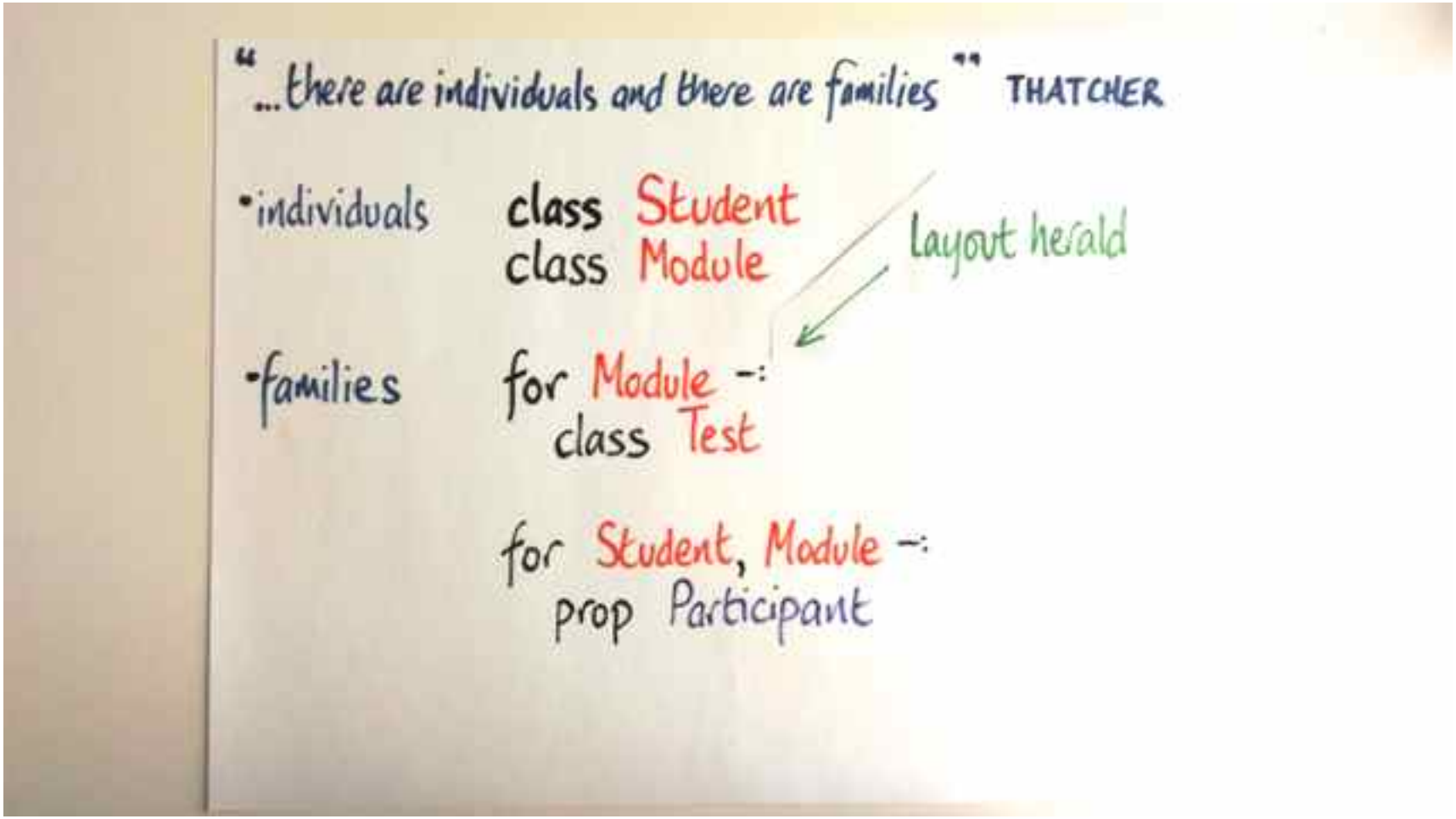
Chop up spreadsheets, too?

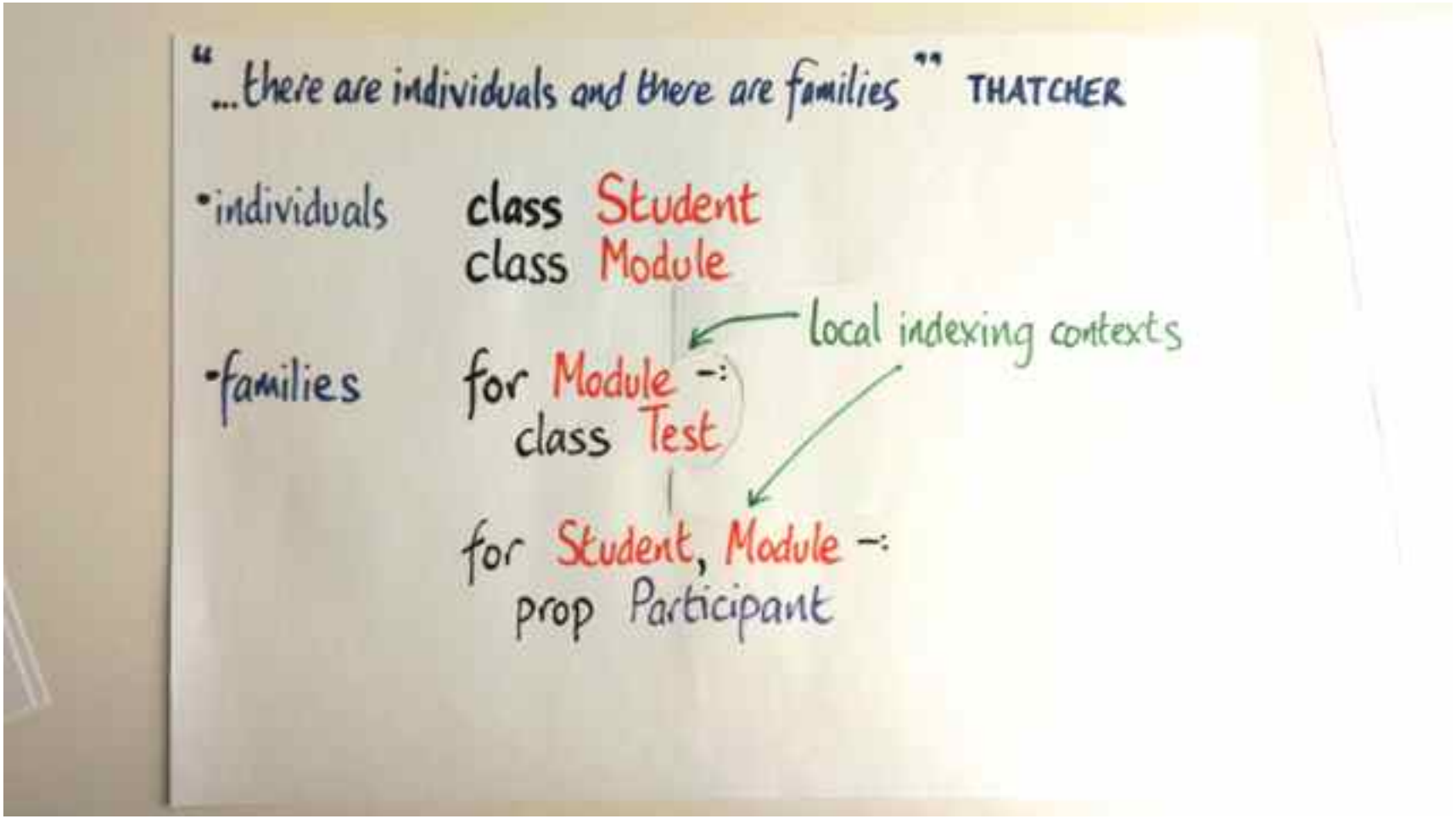

...
Test 3
/20
...
Freda Bloggs
17

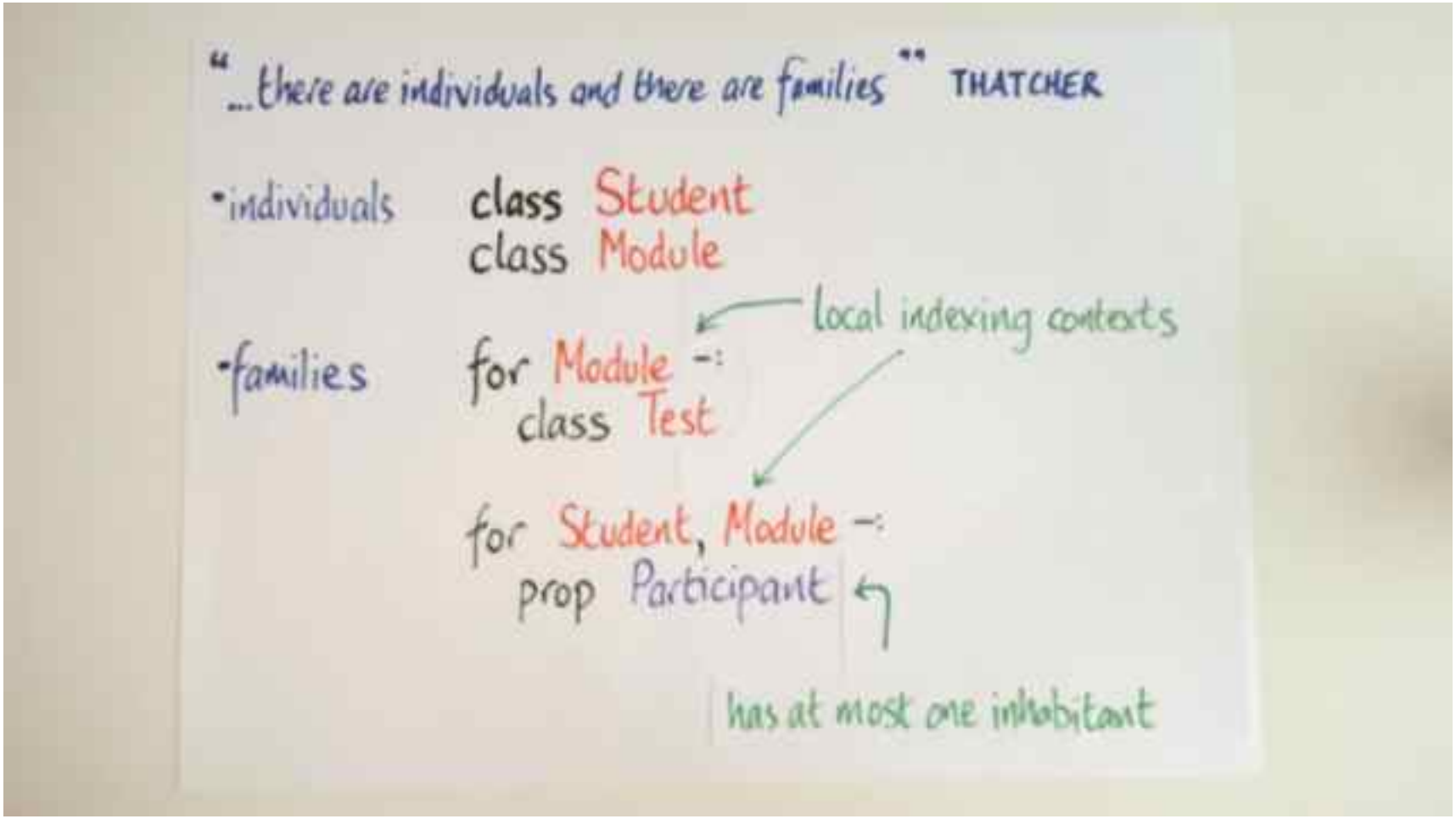

...
Test 3
/20
...
Freda Bloggs
17
to give a meaningful account of the view,
first give a meaningful account of the model

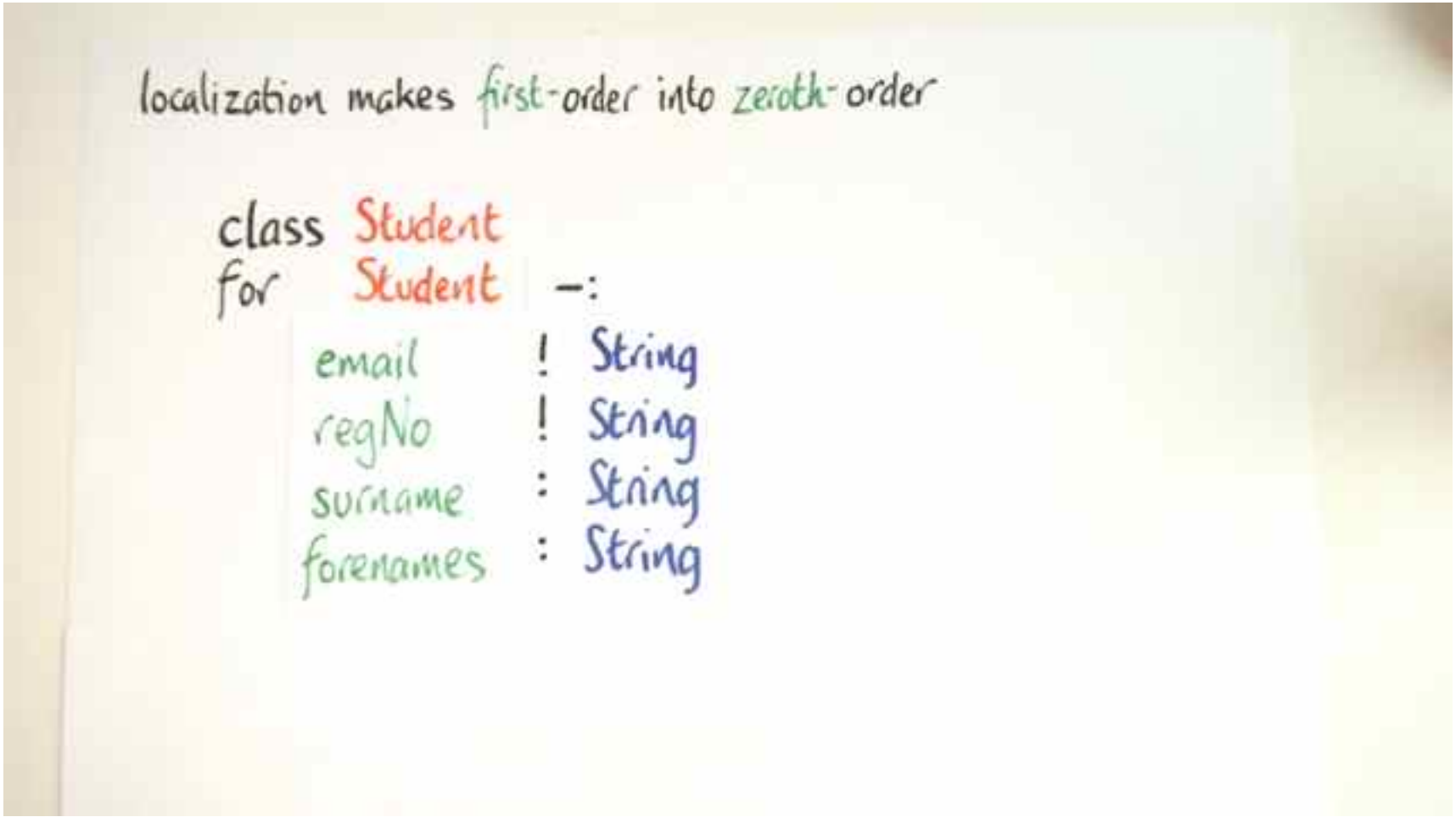

"...there are individuals and there are families" THATCHER
•individuals
class Student
class Module
finite open enumerations

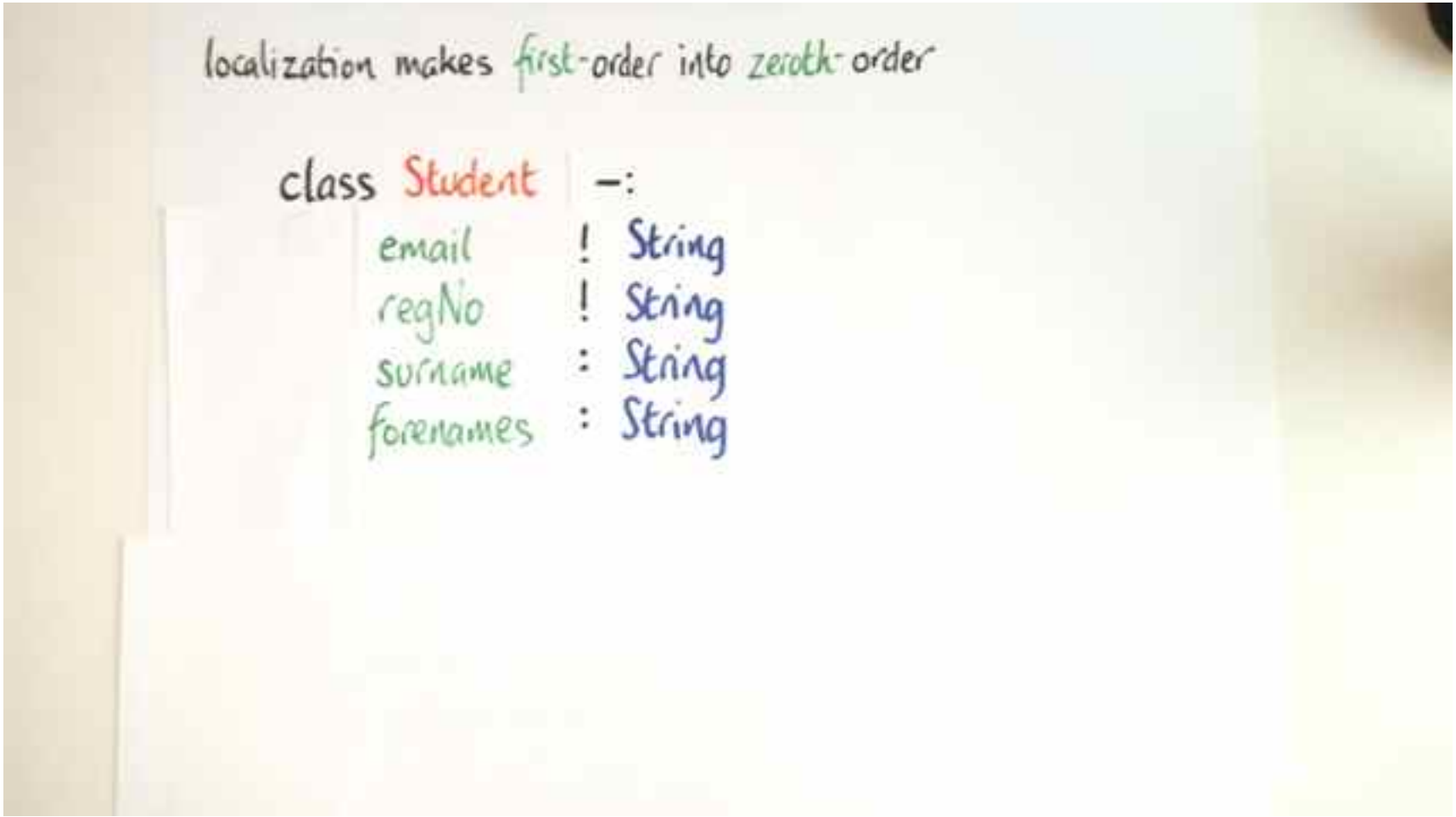

"...there are individuals and there are families" THATCHER
•individuals
class Student
class Module
finite open enumerations
WTF?

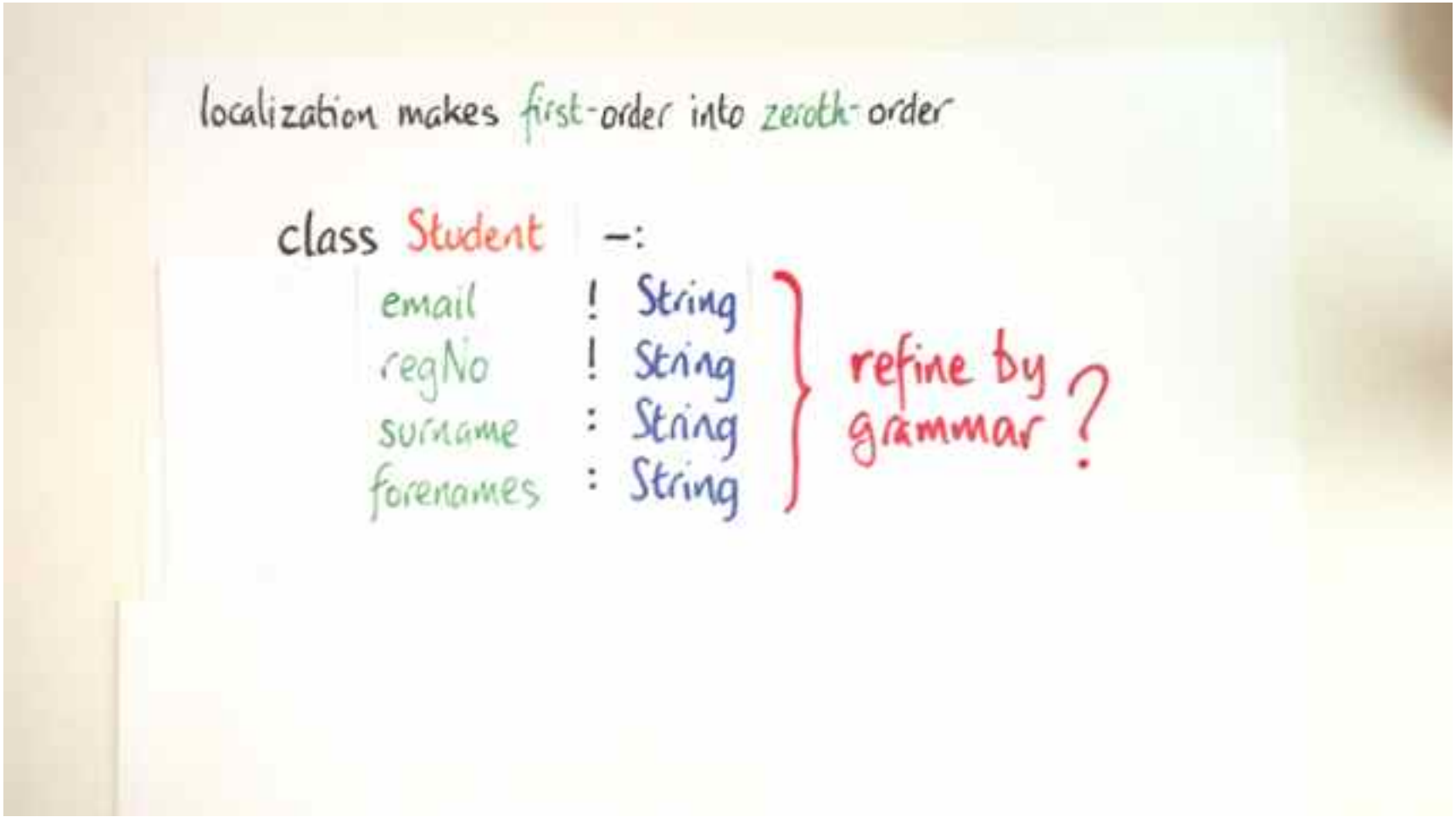

"...there are individuals and there are families" THATCHER
•individuals
class Student
class Module
layout herald
•families
for Module -:
class Test
for Student, Module -:
prop Participant

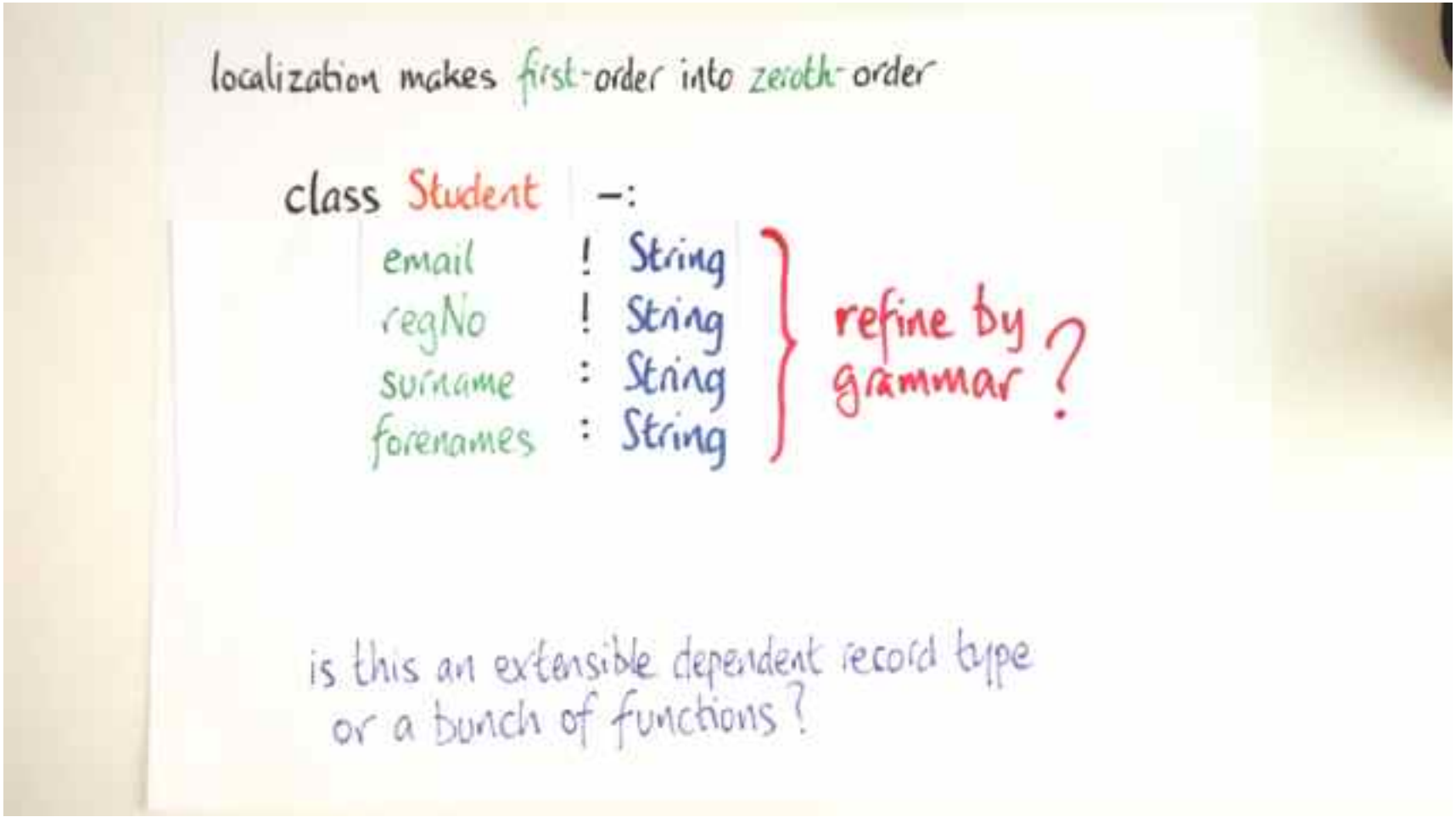

"...there are individuals and there are families" THATCHER
•individuals
class Student
class Module
local indexing contexts
•families
for Module -:
class Test
for Student, Module -:
prop Participant

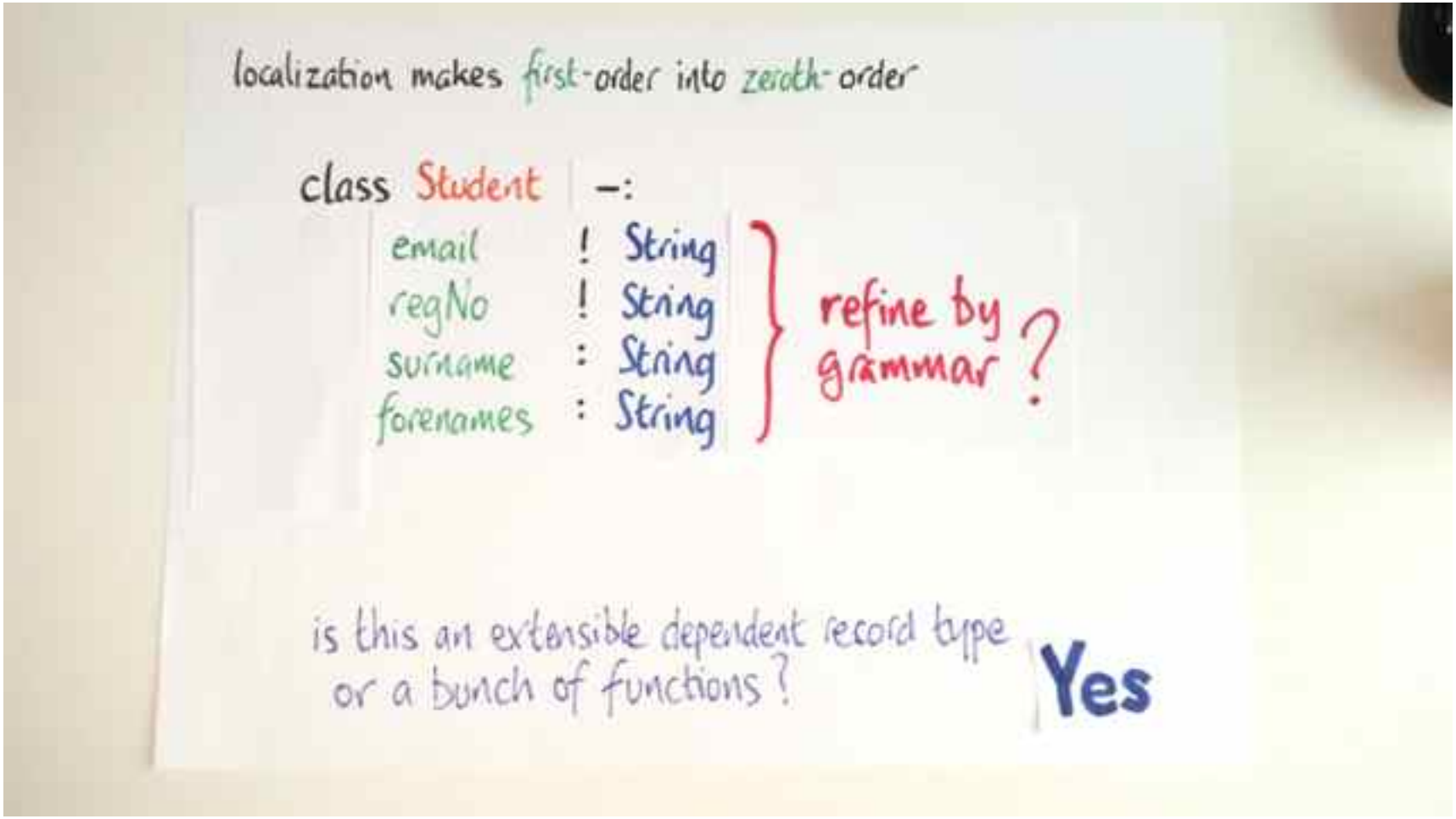

"...there are individuals and there are families" THATCHER
•individuals
class Student
class Module
•families
for Module -:
class Test
local indexing contexts
for Student, Module -:
prop Participant
has at most one inhabitant

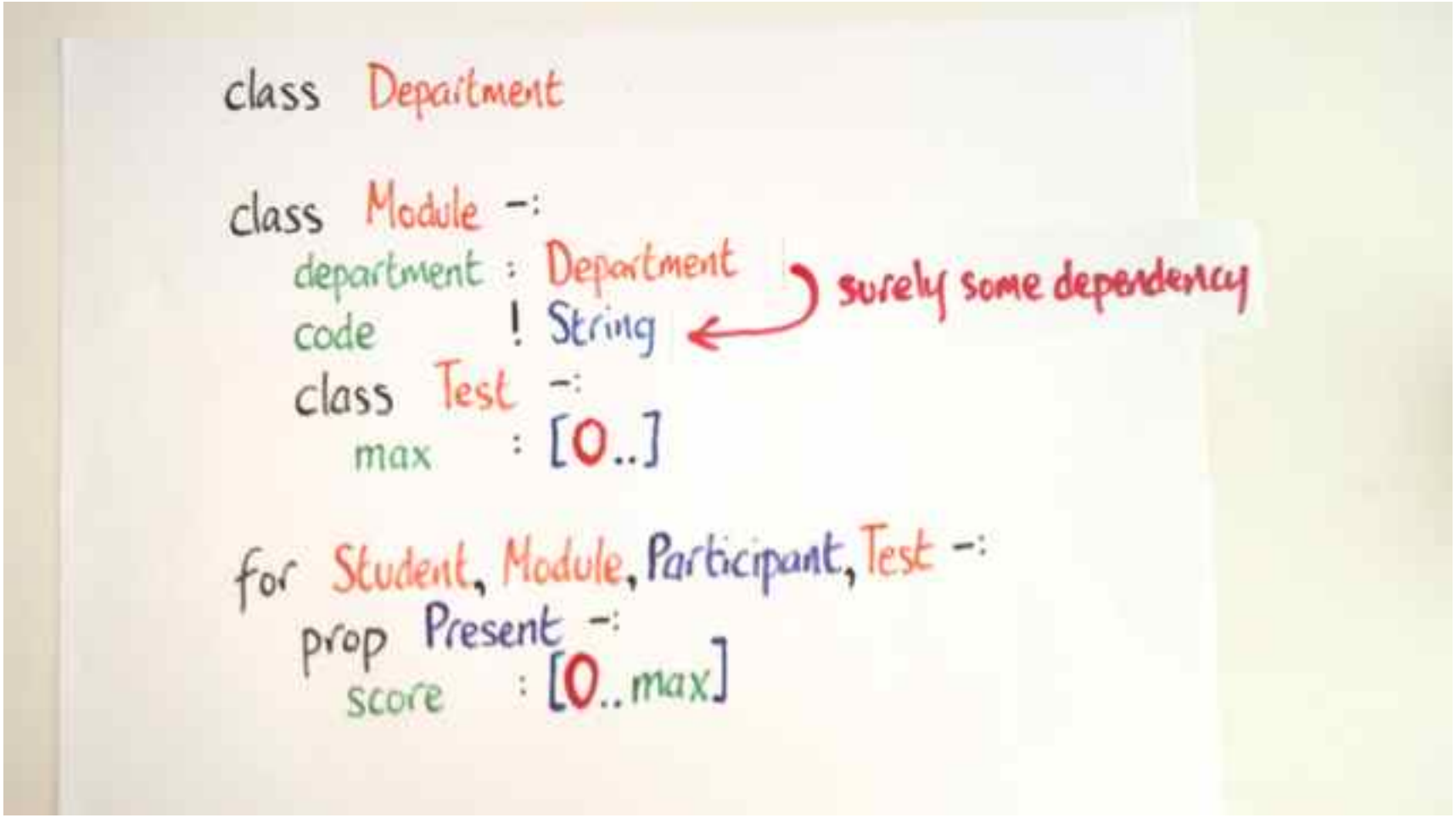

localization makes first-order into zeroth-order
class Student
for Student -:
email ! String
regNo ! String
surname : String
forenames : String

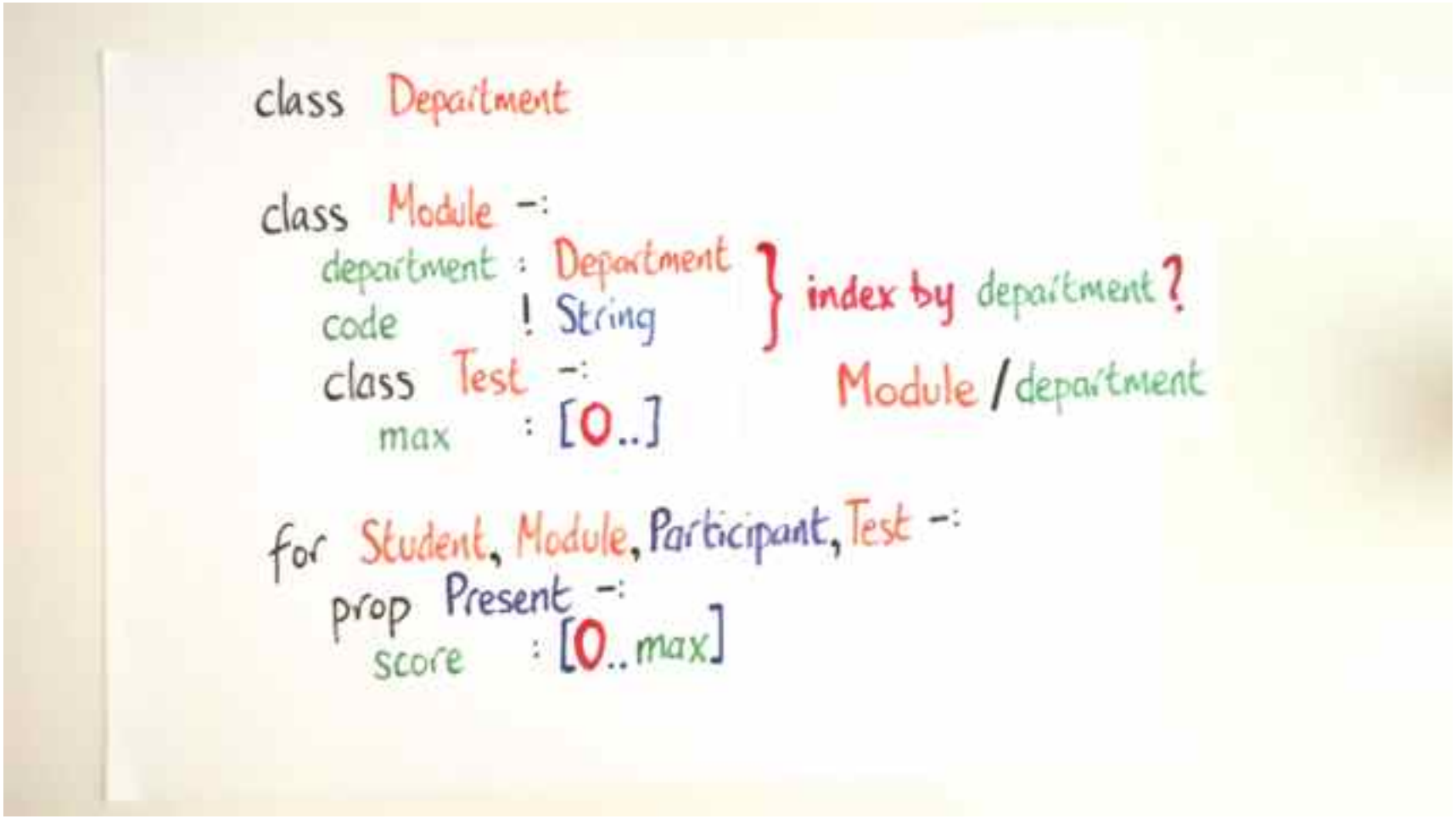

localization makes first-order into zeroth-order
class Student -:
email ! String
regNo ! String
surname : String
forenames : String


localization makes first-order into zeroth-order
class Student -:
email ! String
regNo ! String
surname : String
forenames : String
refine by grammar ?


localization makes first-order into zeroth-order
class Student -:
email ! String
regNo ! String
surname : String
forenames : String
refine by grammar ?
is this an extensible dependent record type or a bunch of functions ?


localization makes first-order into zeroth-order
class Student -:
email ! String
regNo ! String
surname : String
forenames : String
refine by grammar ?
is this an extensible dependent record type or a bunch of functions ?
Yes


class Department
class Module -:
department : Department
code ! String
surely some dependency
class Test -:
max : [0..]
for Student, Module, Participant, Test -:
prop Present -:
score : [0..max]


class Department
class Module -:
department : Department
code ! String
class Test -:
max : [0..]
} index by department?
Module / department
for Student, Module, Participant, Test -:
prop Present -:
score : [0..max]

digression on join

slow! $s{:}\,Student,\ e{:}\,ExamMark,\ s.regNo = e.regNo$

faster? $\exists regNo.\ Student/regNo,\ ExamMark/regNo$

digression on join

slow! s:Student, e:ExamMark, s.regNo = e.regNo } filtered cartesian product zzzz

faster? ∃regNo. Student/regNo, ExamMark/regNo

digression on join

slow! s:Student, e:ExamMark, s.regNo = e.regNo } filtered cartesian product zzzz

faster? ∃regNo. Student/regNo, ExamMark/regNo

fibred cartesian product

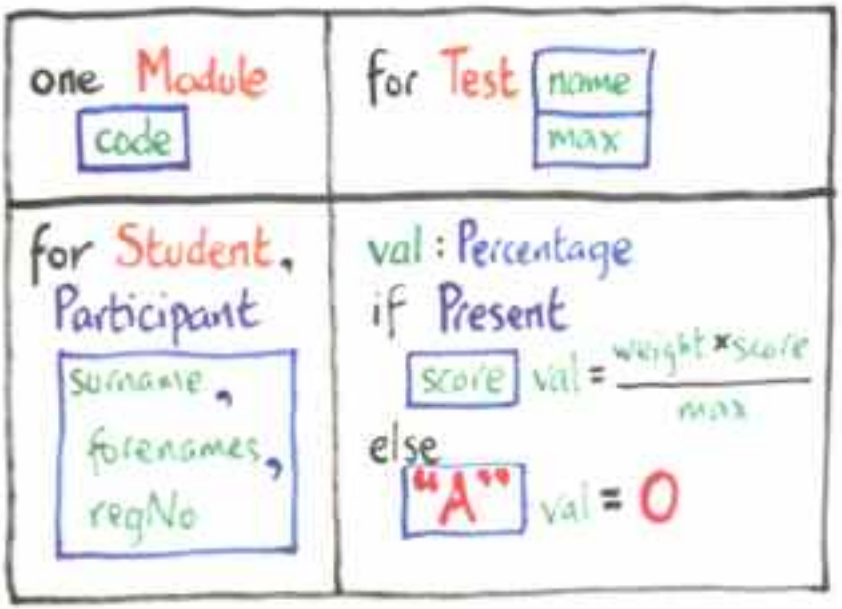
vroom

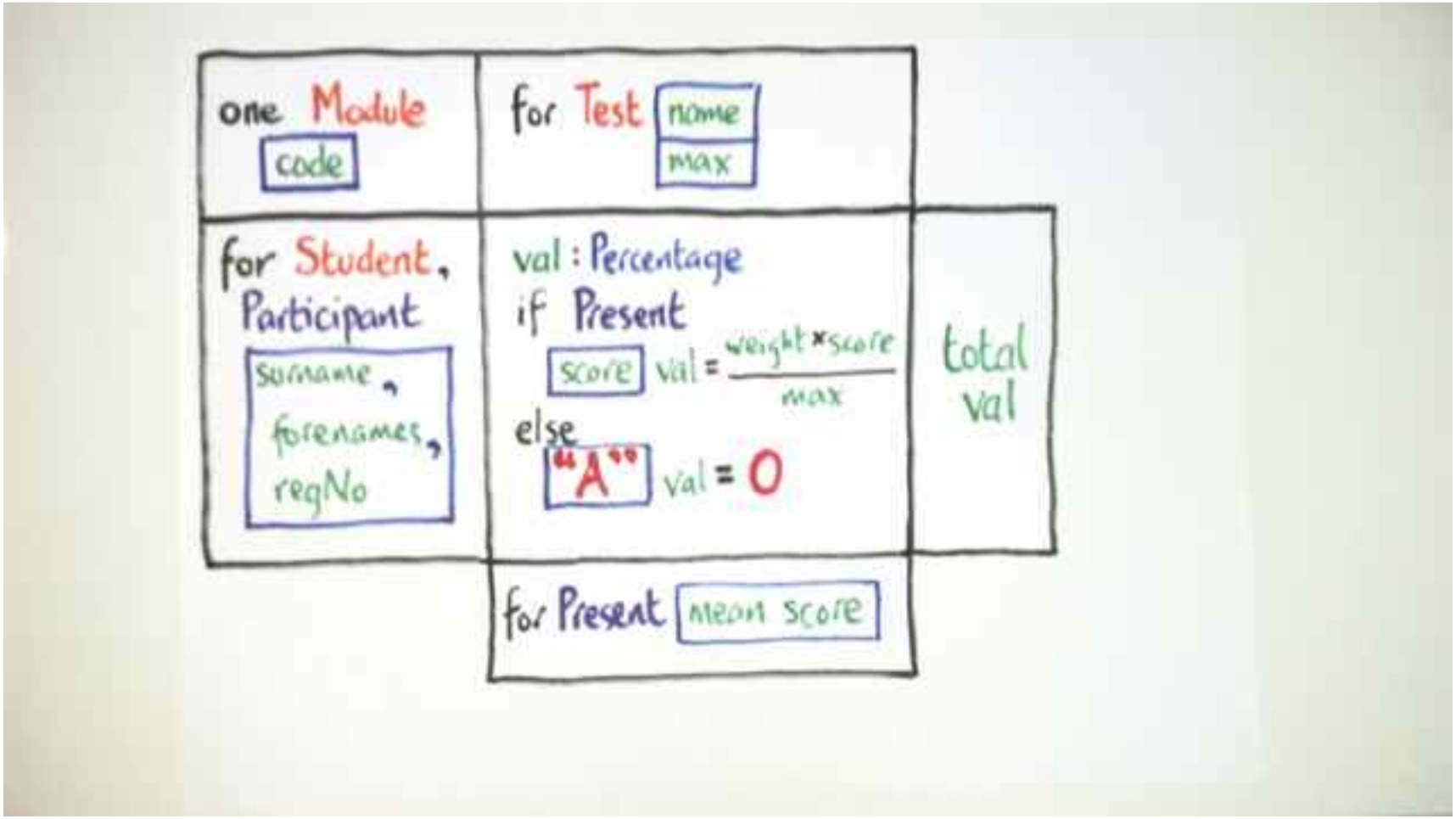
(with discrimination by methods of Henglein, index → key)

your data
a bundle of files, including
your data model
a description of each file's data w.r.t. the model

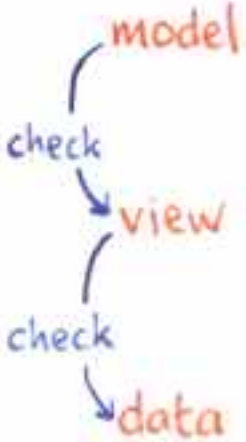
you can (modulo optimism)
audit for conformity and completeness
generate views

your clients can (more optimism)
integrate their model with your model


one Module
code
for Test
name
max
for Student,
Participant
surname,
forenames,
regNo
val : Percentage
if Present
score
val = weight * score / max
else
"A"
val = 0

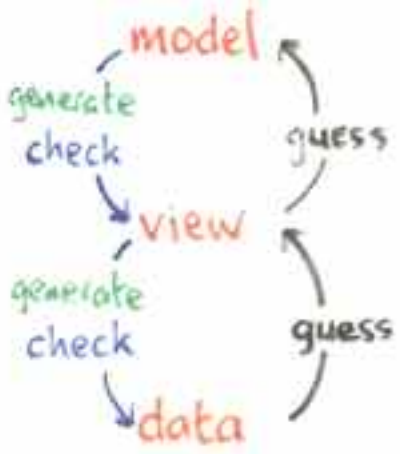

one Module
code
for Test
name
max
for Student, Participant
surname, forenames, regNo
val : Percentage
if Present
score
val = weight * score / max
else
"A"
val = 0
total val
for Present
mean score

Inconclusion

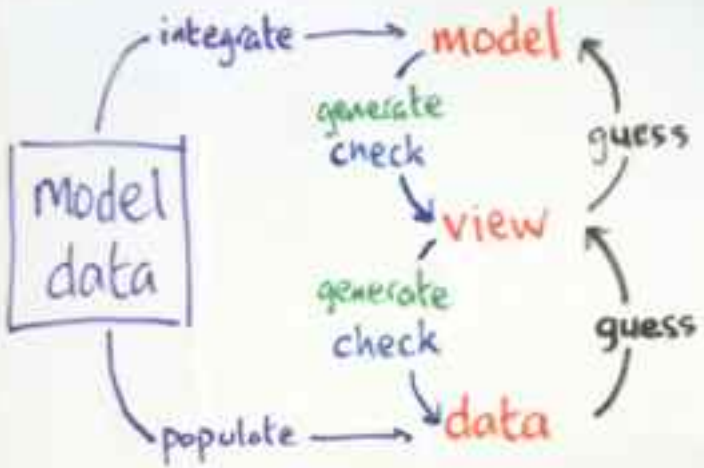


Inconclusion

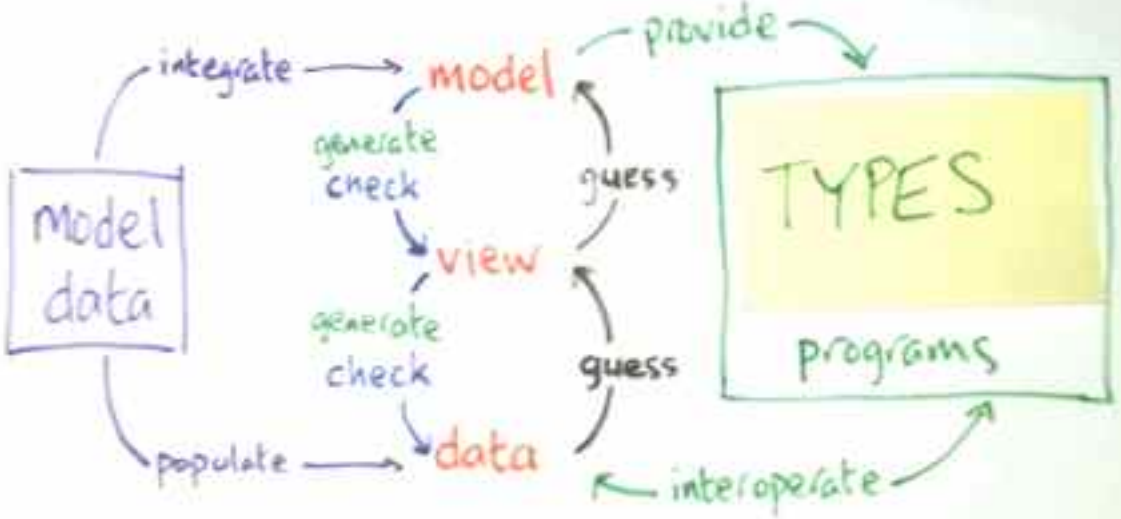
Inconclusion

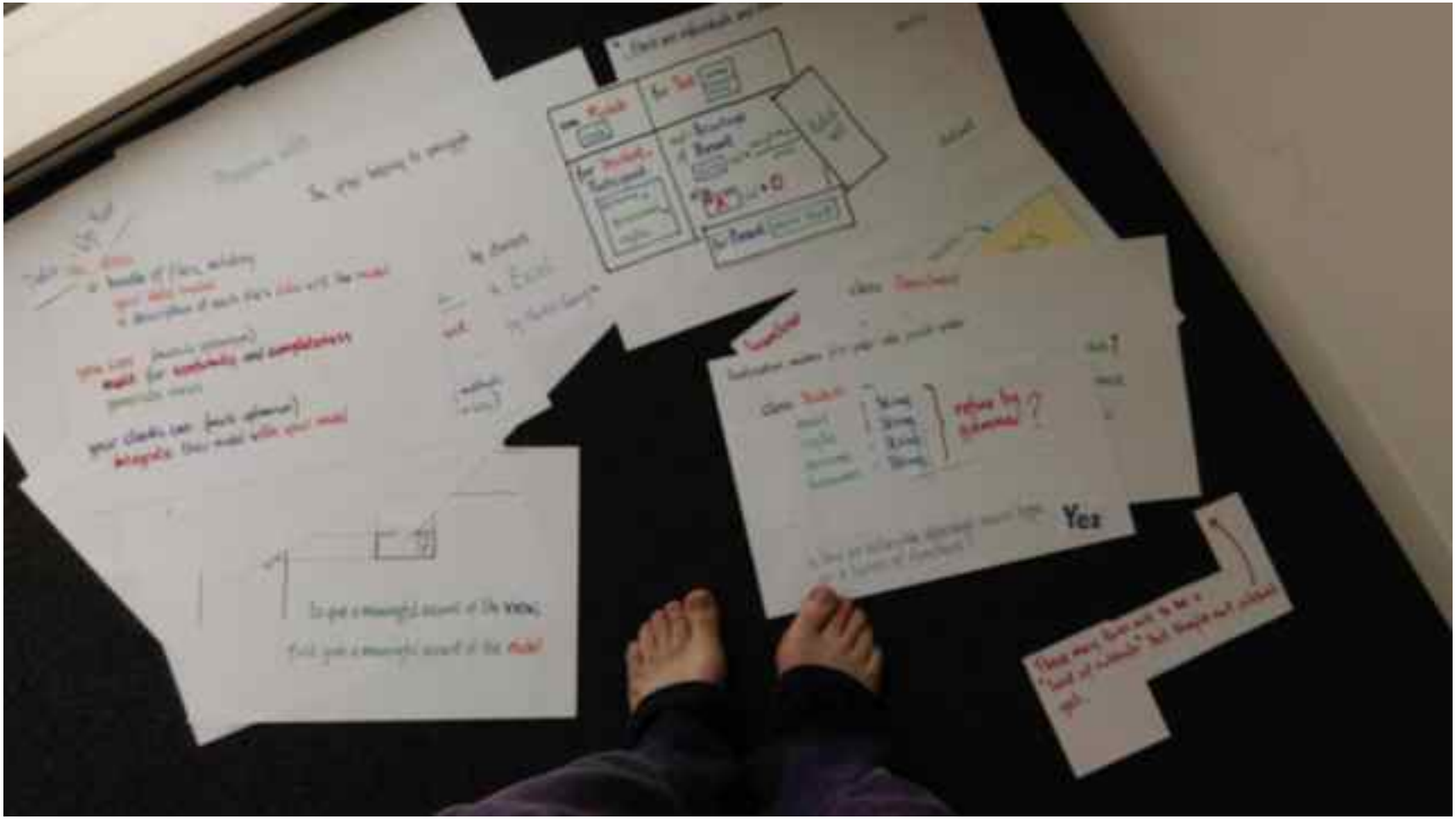


Inconclusion



Inconclusion




Yes