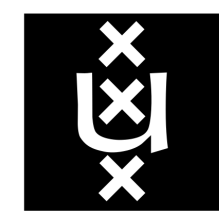


A principled approach to REPLs*

*Read-eval-print-loops, consoles, interactive shells, notebooks, command-lines



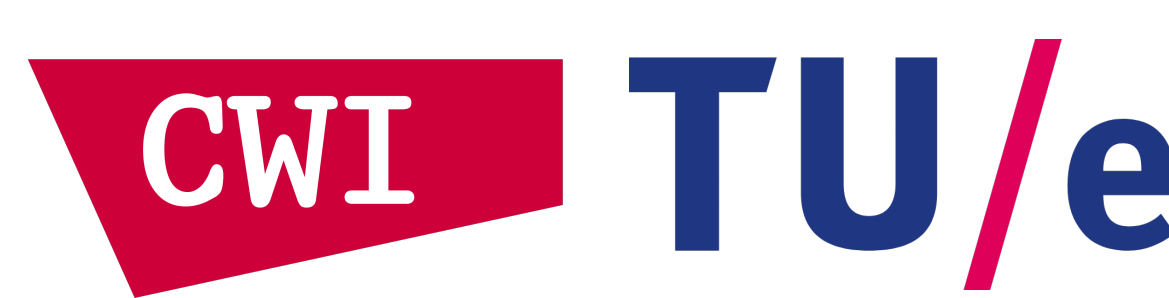
Thomas van Binsbergen



UNIVERSITEIT VAN AMSTERDAM



Mauricio Verano



Tijs van der Storm



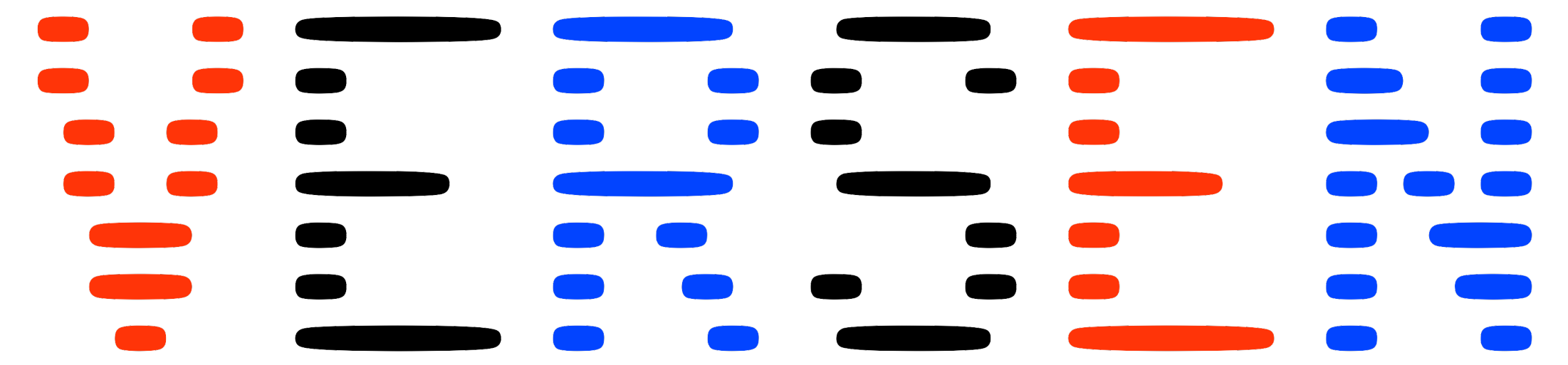
university of
 groningen

Joint work with Pierre Jeanjean, Benoit Combemale, Olivier Barais (INRIA, University of Rennes)

Problem: REPLs are popular tools, but their semantics is not well-defined

Solution: see REPLs as a (modular) *language extension*

How: define a *sequential* operator “;” and *compose* semantics.

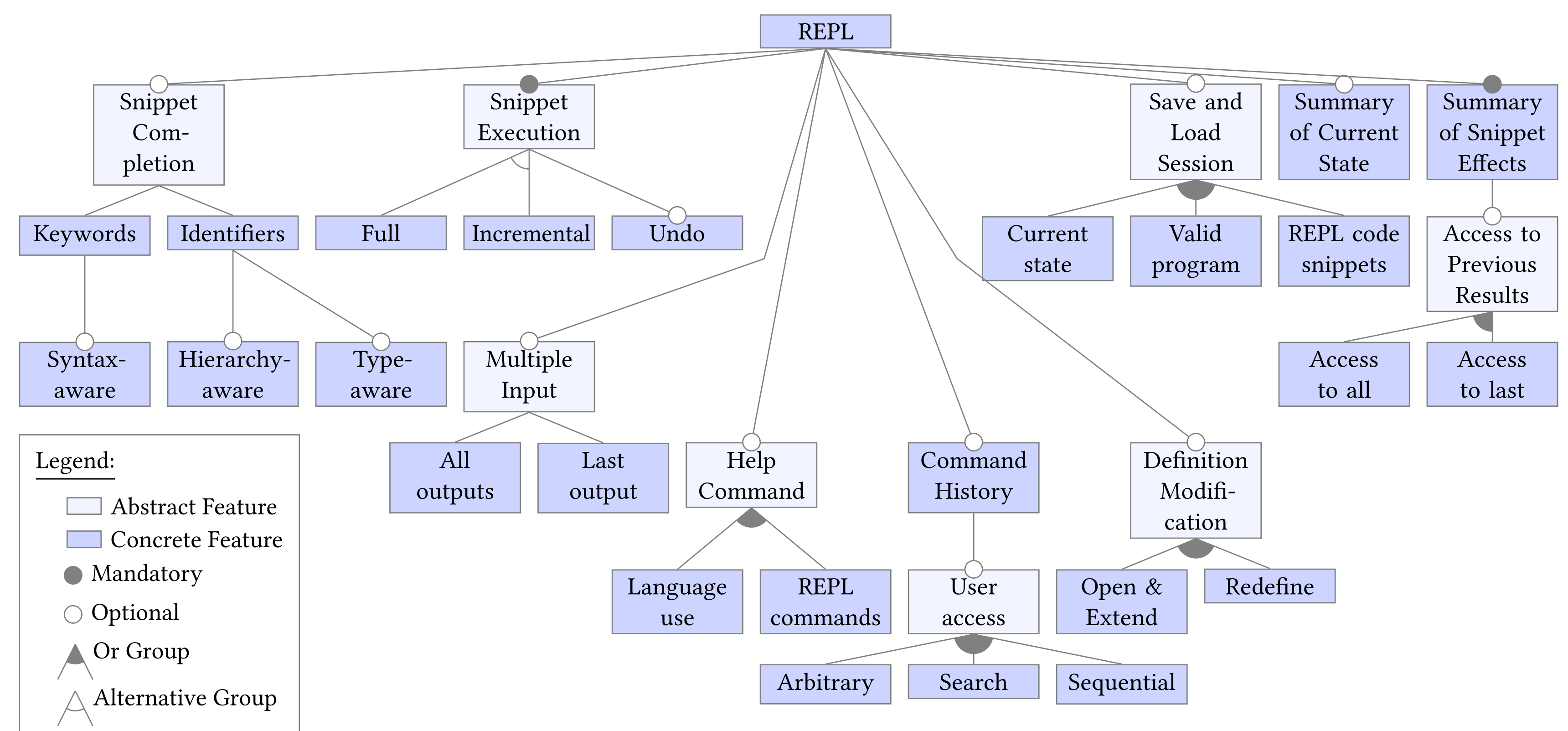


REPLs: old but popular

Very diverse design space:

U:	Type 2+2.	
J:	2+2 =	4
U:	Set x=3.	
J:	Type x.	
U:	x =	3
J:	Type x+2, x-2, 2*x, x/2, x*2.	
U:	x+2 =	5
J:	x-2 =	1
U:	2*x =	6
J:	x/2 =	1.5
U:	x*2 =	9
U:	Type [(x-5)*3+4]*2-15]*3+10.	
J:	[(x-5)*3+4]*2-15]*3+10 =	25

JOSS (1964)



Mar 2022	Mar 2021	Change	Programming Language
1	3	▲	Python
2	1	▼	C
3	2	▼	Java
4	4		C++
5	5		C#

Sequential languages:

“A language is *sequential* if the concatenation of two programs is again a program”



$$[p_1 \circ p_2] = [p_2] \circ [p_1]$$

Non-seq stack language:

```
syntax Prog = Op* Print;  
syntax Op = Num | "+" | "dup";  
syntax Print = "print";  
alias Stack = list[int];  
int eval((Prog)`<Op* ops> print`)
```

Prog “.” Prog ≠ Prog!

Sequentialize:

$$p_1 \circ p_2$$

```
syntax Cmd = Op | Print | assoc Cmd Cmd ;
```

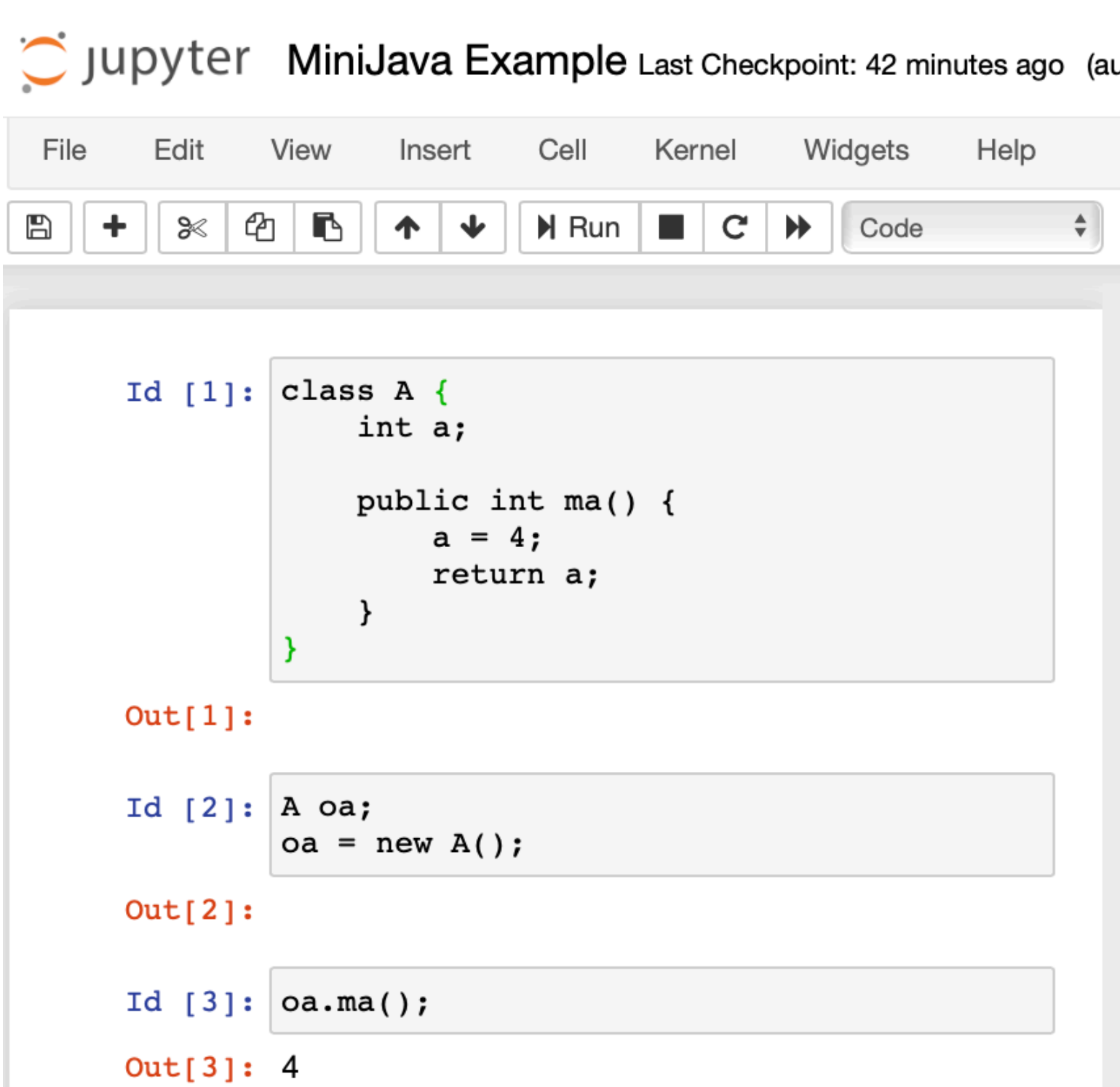
```
Conf eval((Cmd)`<Cmd c1> <Cmd c2>`, Conf c)  
= eval(c2, eval(c1, c));
```

$$[p_2] \circ [p_1]$$

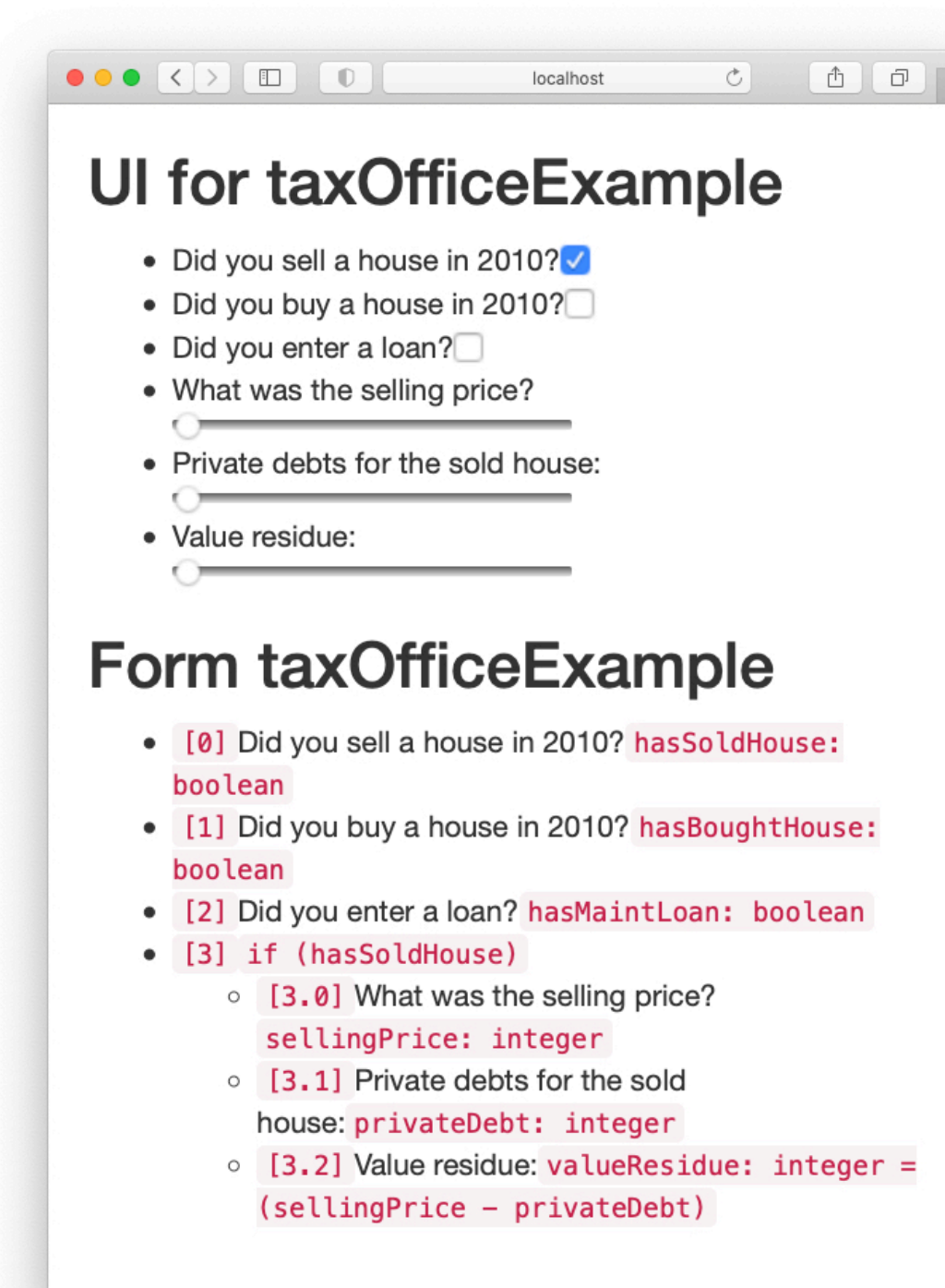
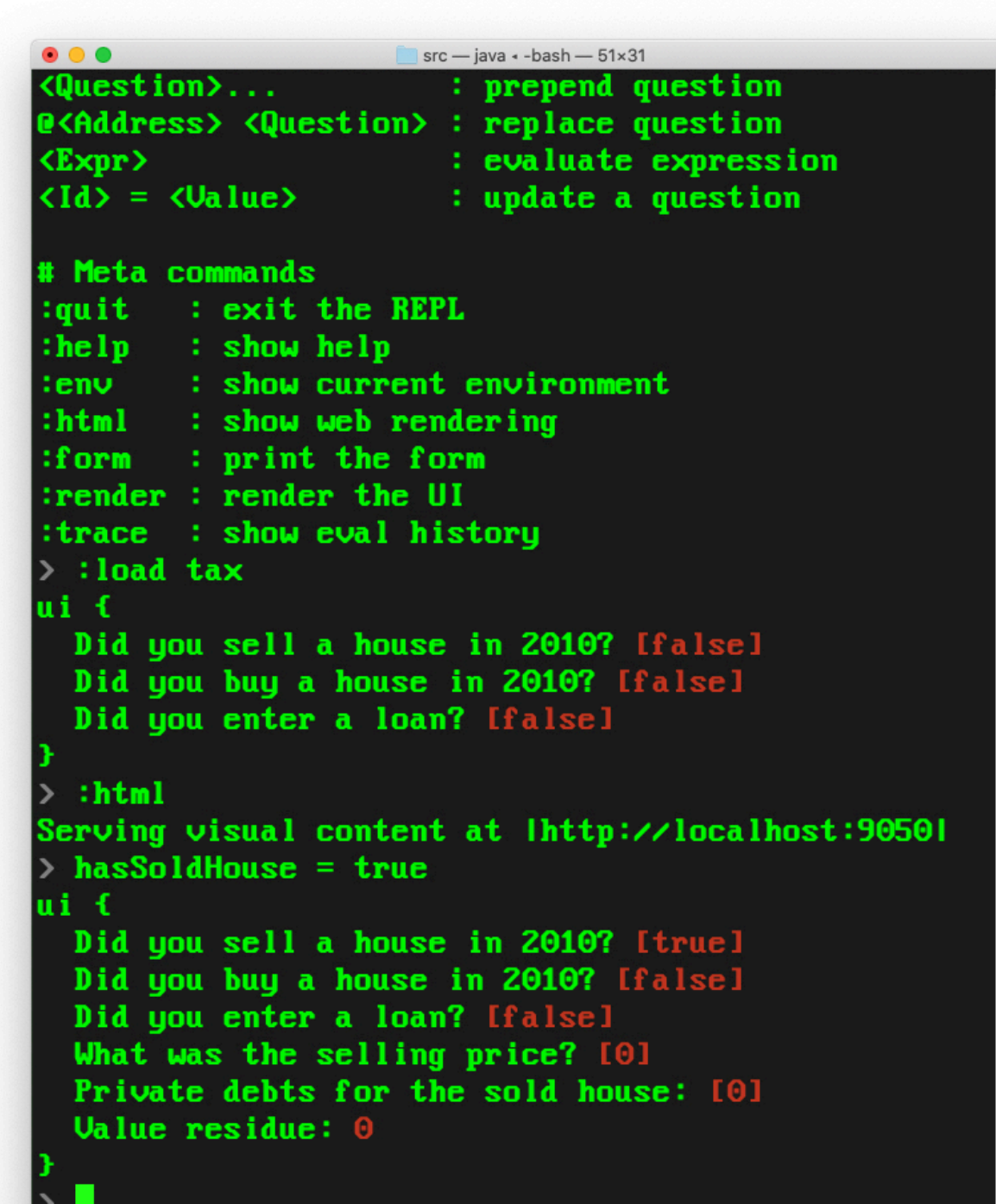
rascal-mpl.org



MiniJava Notebook



QL: questionnaire DSL



eFLINT: norms DSL

