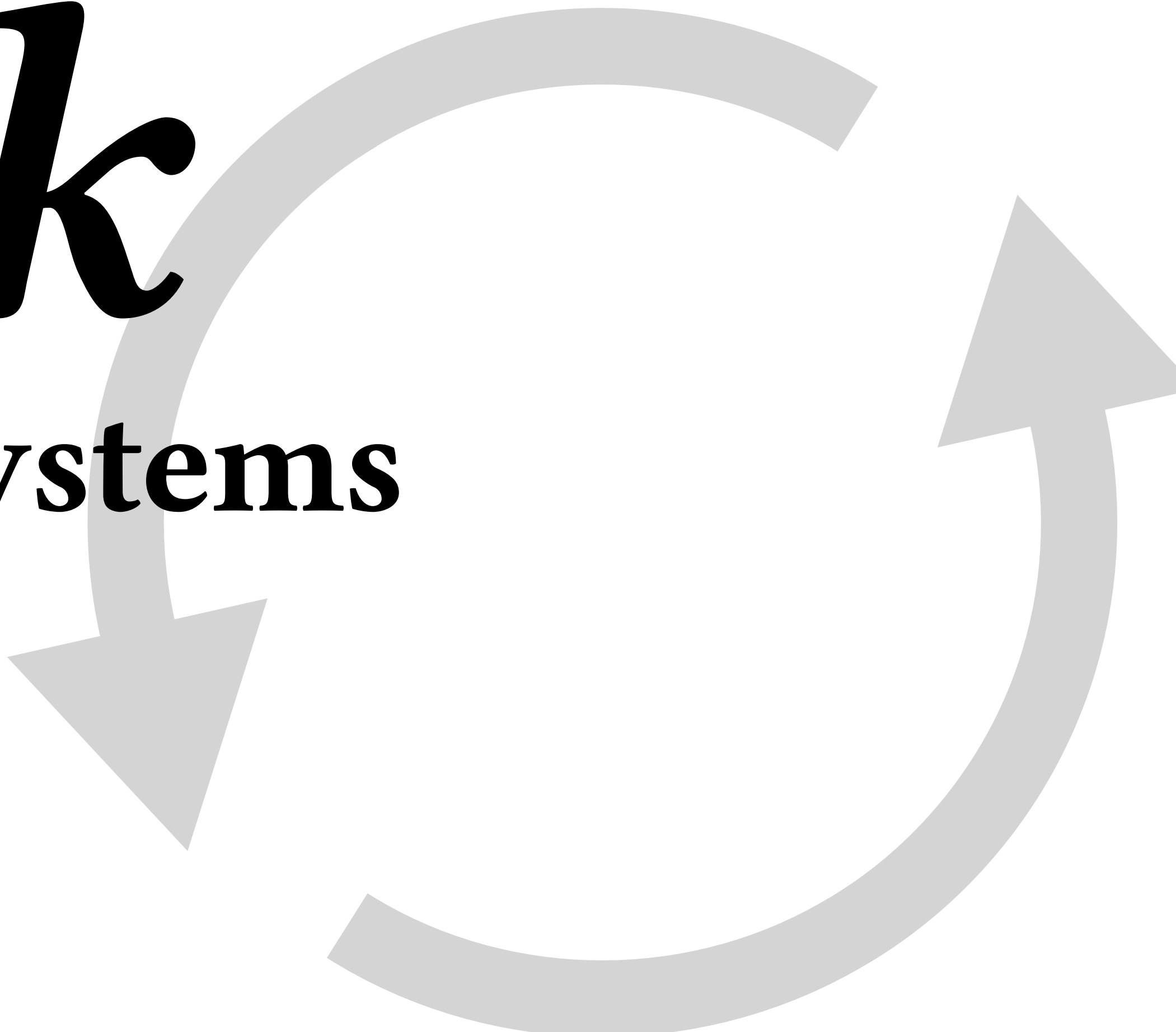


Technical Dimensions of  
*Feedback*  
in Live Programming Systems



Josh Horowitz, LIVE 2024



You can make arcs with the procedure `arcleft` and `arcright`.

```
arcleft input size degrees
repeat degrees
  ray repeat 2
    arcleft r 90
    arcright r 90
  repeat 9
    ray
    right 160
```

Here are some procedures that draw pictures:

```
flower S: 1.4
slinky R: 2.6
```

Try running some of the following commands to draw pictures in the graphics box above.

```
sun R: 0.7
```

Subtext

```
fibonacci
in: 1
out: 0
>= 2
```

```
fibonacci
in: 2
out: 0
>= 2
```

```
fibonacci
in: 1
out: 1
>= 2
```

```
fibonacci
in: 2
out: 1
>= 2
```

```
fibonacci
in: 3
out: 1
>= 2
```

```
fibonacci
in: 4
out: 2
>= 2
```

```
fibonacci
in: 5
out: 3
>= 2
```

```
fibonacci
in: 6
out: 5
>= 2
```

```
fibonacci
in: 7
out: 8
>= 2
```

```
fibonacci
in: 8
out: 13
>= 2
```

```
fibonacci
in: 9
out: 21
>= 2
```

```
fibonacci
in: 10
out: 34
>= 2
```

```
var i = 0;
while (i < 20) {
  var scaleFactor = 1 + (20 - i)/20;
  resetMatrix();
  scale(scaleFactor);
  rotate(i * 6);
  fill(i * 30, i * 18, 0);
  triangle(0, 0, 100, -20, 95, 40); Draw a triangle.
  i += 1;
}
```

# What is "liveness"?

draggable

```
on('mousedown', this)
on('mouseup', this)
no_drag
```

Copies:

Add Field

prototypes (div)	dom.div	x	y	fill
x: 313	0	0	'black'	
y: 763	0	'black'		
fill: 'black'				

System Browser

- Collections-Sequence
- Collections-Text
- Collections-Array
- Collections-Stream
- Collections-Support
- Graphics-Primitives
- Graphics-Display
- Graphics-Media
- Graphics-Paths

accessing copying adding removing enumerating private

```
collect:
do:
do:andBetweenDo:
promoteFirstSuchT
reverse
reverseDo:
select:
Form Editor
```

collect: aBlock

"Evaluate aBlock with each of my elements as the argument. Collecting resulting values into a collection that is like me. Answer with collection. Override superclass in order to use add:, not at:put:."

```
| newCollection |
newCollection + self species new.
self do: [each | newCollection add: (aBlock value: each)].
+newCollection
```

User Interrupt

```
Paragraph>>characterBlockAtPoint:
Paragraph>>mouseSelect:to:
CodeController(ParagraphEditor)>>processRedButton
CodeController(ParagraphEditor)>>processMouseButtons
CodeController(ParagraphEditor)>>controlActivity
CodeController(Controller)>>controlLoop
```

controlActivity

```
self scrollBarContainsCursor
ifTrue:
[ self scroll ]
ifFalse:
self scrollBarContainsCursor
```

File List

Fig.1

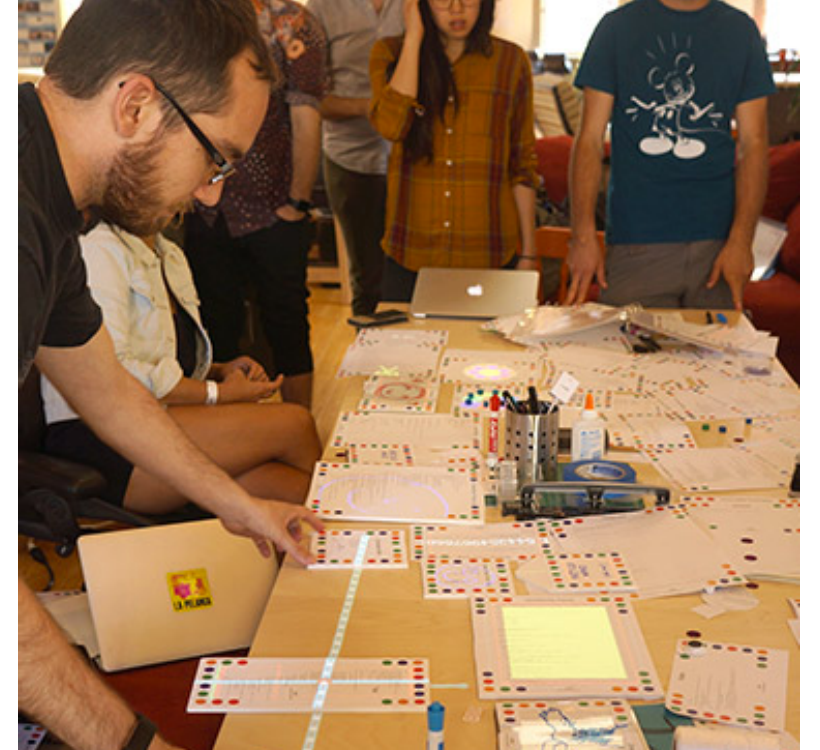
```
C11 (L) TOTAL
```

A	B	C	D
ITEM	NO.	UNIT	COST
MUCK RAKE	43	12.95	556.05
BUZZ CUT	1	101.00	101.00
TOE TONER	25	48.95	1248.75
EYE SNUFF	2	4.95	9.90
SUBTOTAL			13155.50
9.75% TAX			1282.66
TOTAL			14438.16

Move frame so frame's bottom-left meets bar's bottom-right.

Repeat from 1 to 4 columns:

- Draw bar from frame's bottom-left to frame's top-right.
- Scale bar's height around bar's bottom.
- Move frame so frame's bottom-left meets bar's bottom-right.
- Move bar's right - 4 px horizontally, 0 px vertically.



Main UI Loop

Set DIO Values: Value Change

add

Set DIO Values

Set Error

Master Stop

Task One - Bell State

Remember that you may consult the reference sheet to answer any questions you may have regarding how to describe new components, such as the control box.

1

## **The *Feedback Definition* of Liveness:**

Live programming environments provide programmers with continuous **feedback** about a program's dynamic behavior as it is being edited.

# The *Feedback Definition* of Liveness

Live programming environments provide programmers with continuous **feedback** about a program's dynamic behavior as it is being edited.

*Tanimoto 1990*: “Visual programming systems can be classified according to the degree to which they present ‘live’ **feedback** to the programmer.”

*Burnett et al. 1998*: “Tanimoto coined the term ‘liveness,’ which categorizes the immediacy of semantic **feedback** that is automatically provided during programming.”

*Rein et al. 2018*: “[Liveness] seems to be used when describing programming tools which provide immediate **feedback** on the dynamic behavior of a program even while programming.”

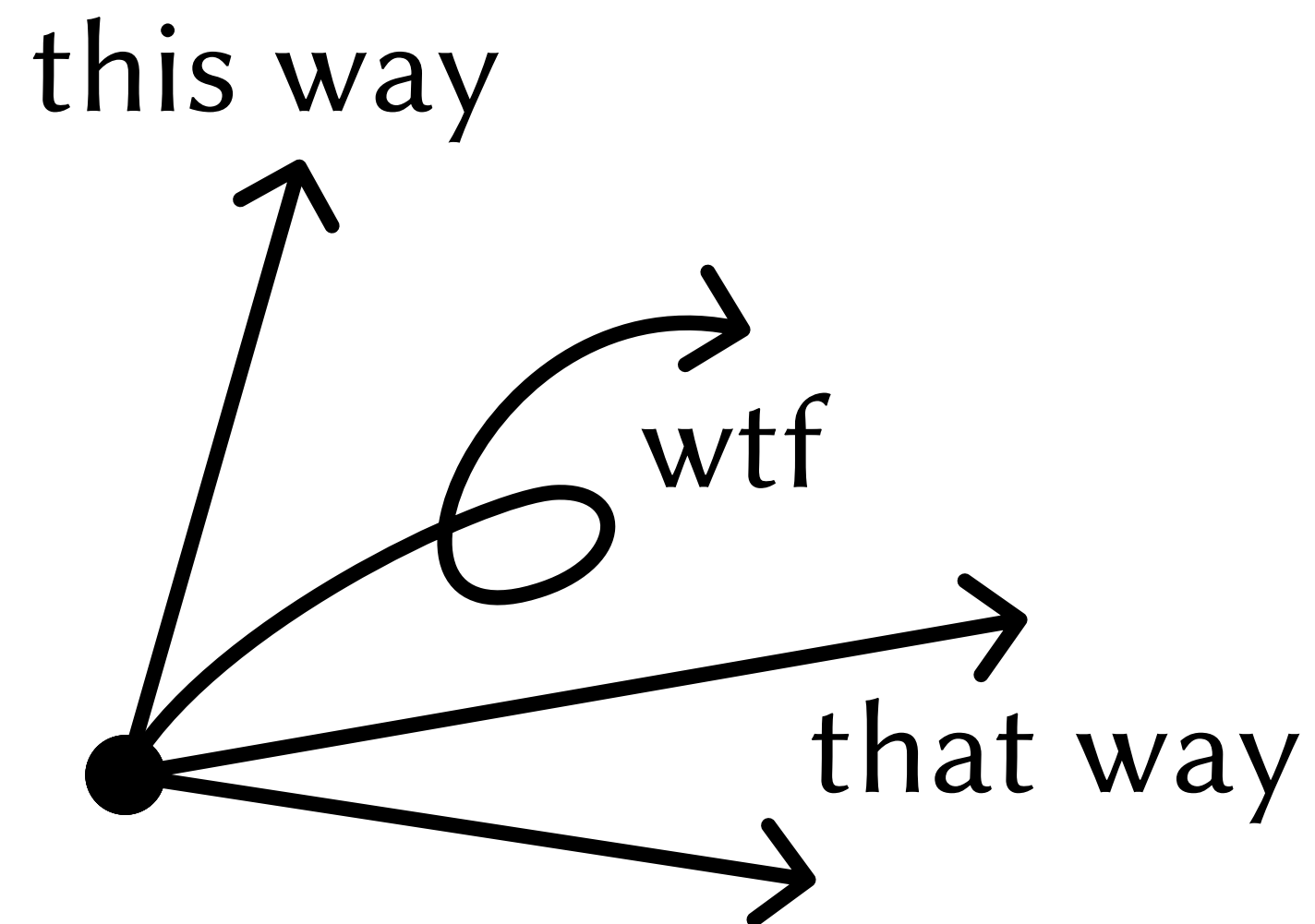
*Omar et al. 2018*: “Live programming environments aim to provide programmers ... with continuous **feedback** about a program's dynamic behavior as it is being edited.”

*Lerner 2020*: “Live programming is a coding regime in which immediate **feedback** is provided to the programmer each time the program is modified.”

*LIVE 2024*: “Live programming systems give the programmer immediate **feedback** on the output of a program as it is being edited.”

# The *Feedback Definition* of Liveness


Live programming environments provide programmers with continuous **feedback** about a program's dynamic behavior as it is being edited.



# Technical Dimensions of Programming Systems

Jakubovic  
Edwards  
Petricek  
2023

## Interaction

 **Interaction**

How do users manifest their ideas, evaluate the result, and generate new ideas in response?


**Dimensions**

What are the gulfs of execution and evaluation and how are they related?  
→ Dimension: Feedback loops

Which sets of feedback loops only occur together?  
→ Dimension: Modes of interaction

How do we go from abstractions to concrete examples and vice versa?  
→ Dimension: Abstraction construction

## Notation

 **Notation**

How are the different textual and visual programming notations related?

**Dimensions**

What notations are used to program the system and how are they related?  
→ Dimension: Notational structure


What is the connection between what a user sees and what a computer program sees?  
→ Dimension: Surface and internal notation

Is one notation more important than others?  
→ Dimension: Primary and secondary notations

Do similar expressions encode similar programs?  
→ Dimension: Expression geography

Does the notation use a small or a large number of basic concepts?  
→ Dimension: Uniformity of notations

## Conceptual structure

 **Conceptual structure**

How is meaning constructed? How are internal and external incentives balanced?

**Dimensions**


Does the system present as elegantly designed or pragmatically improvised?  
→ Dimension: Conceptual integrity versus openness

What are the primitives? How can they be combined to achieve novel behaviors?  
→ Dimension: Composability

Which wheels do users not need to reinvent?  
→ Dimension: Convenience

How much is common structure explicitly marked as such?  
→ Dimension: Commonality

## Customizability

 **Customizability**

Once a program exists in the system, how can it be extended and modified?


**Dimensions**

Must we customize running programs differently to inert ones? Do these changes last beyond termination?  
→ Dimension: Staging of customization

Which portions of the system's state can be referenced and transferred to/from it? How far can the system's behavior be changed by adding expressions?  
→ Dimension: Addressing and externalizability

How far can the system's behavior be changed from within?  
→ Dimension: Self-sustainability

## Complexity

 **Complexity**


How does the system structure complexity and what level of detail is required?

**Dimensions**

What programming details are hidden in reusable components and how?  
→ Dimension: Factoring of complexity

What part of program logic does not need to be explicitly specified?  
→ Dimension: Level of automation

## Errors

 **Errors**


What does the system consider to be an error? How are they prevented and handled?

**Dimensions**

What errors can be detected in which feedback loops, and how?  
→ Dimension: Error detection

How does the system respond when an error is detected?  
→ Dimension: Error response

## Adoptability

 **Adoptability**

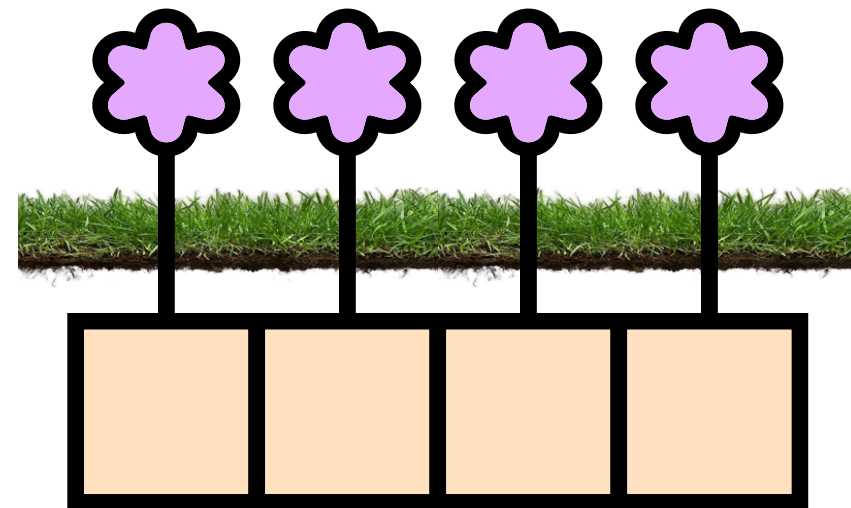
How does the system facilitate or obstruct adoption by both individuals and communities?

**Dimensions**

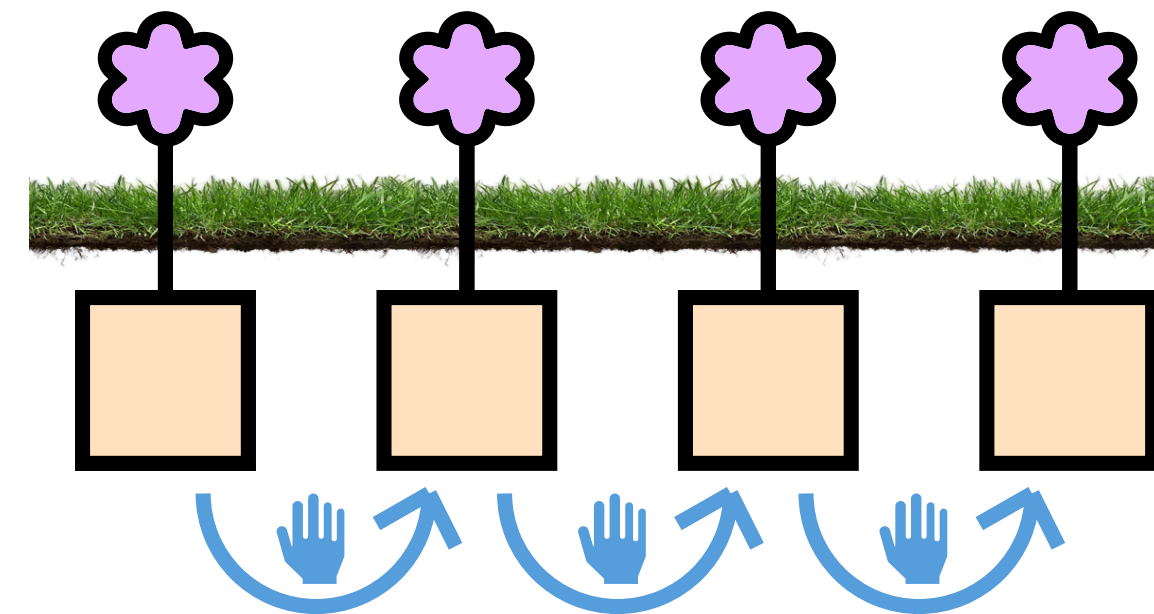
What is the attitude towards the learning curve and what is the target audience?  
→ Dimension: Learnability

What are the social and economic factors that make the system the way it is?  
→ Dimension: Sociability

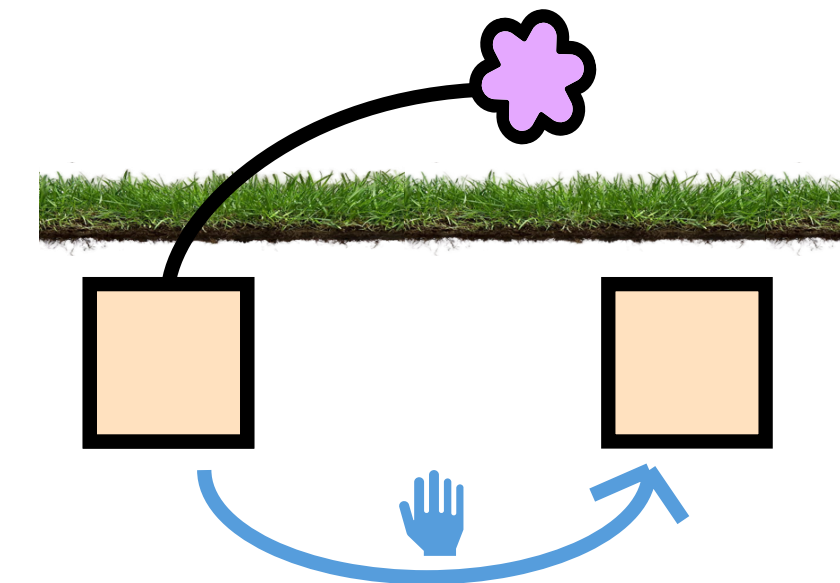
# Technical Dimensions of *Feedback* in Live Programming Systems



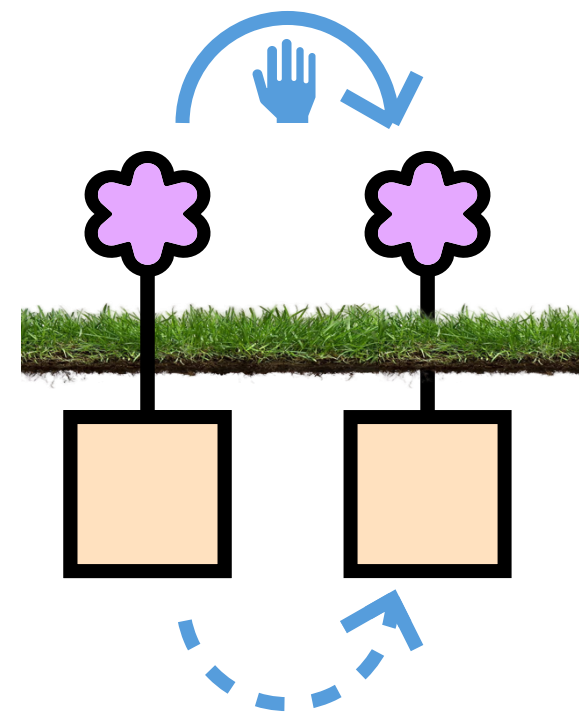
**Granularity**



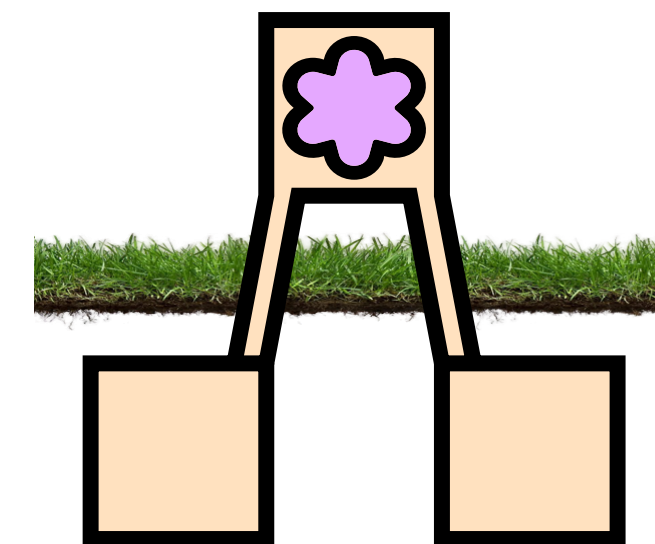
**Reactivity**



**Velocity**

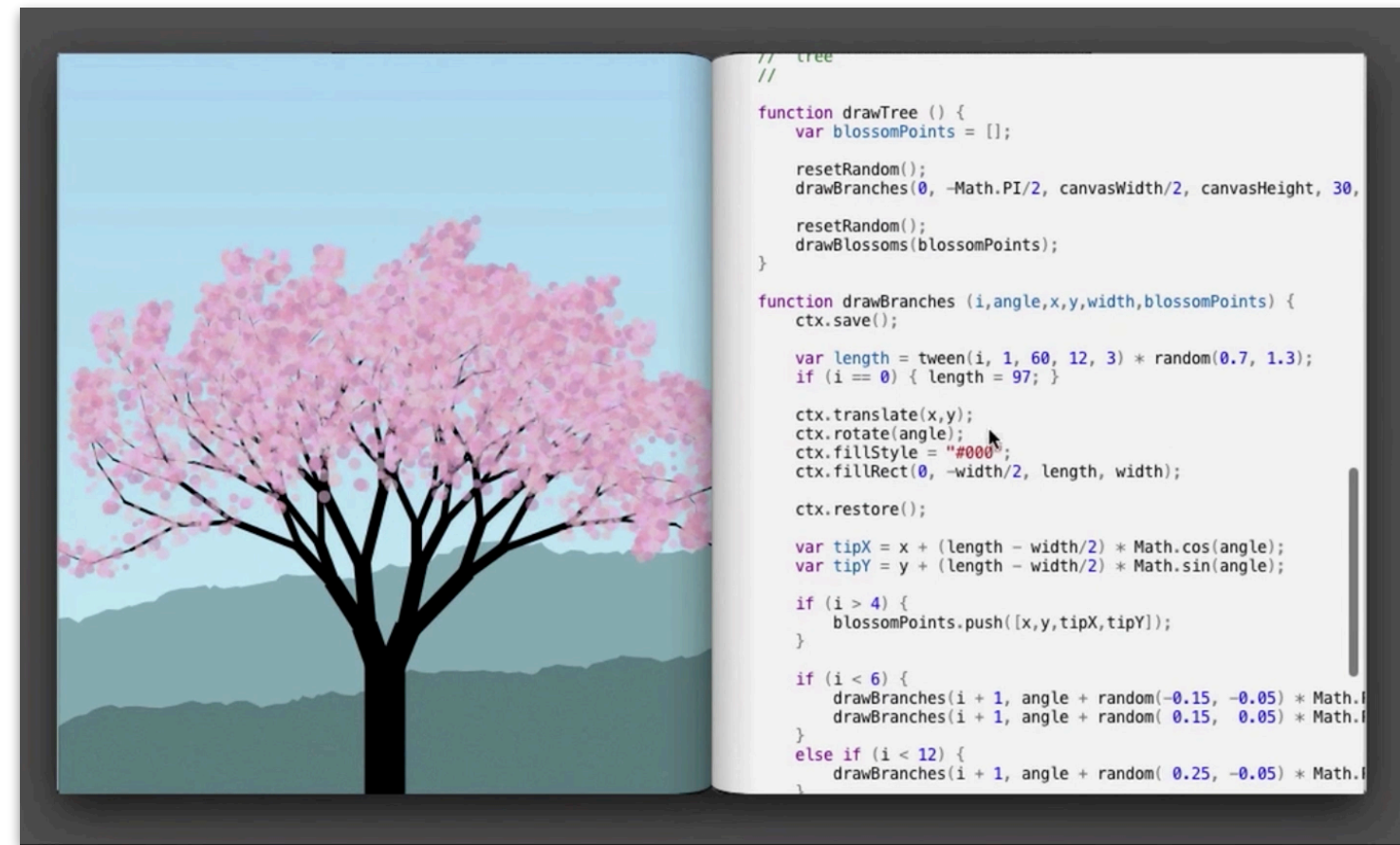


**Bidirectionality**

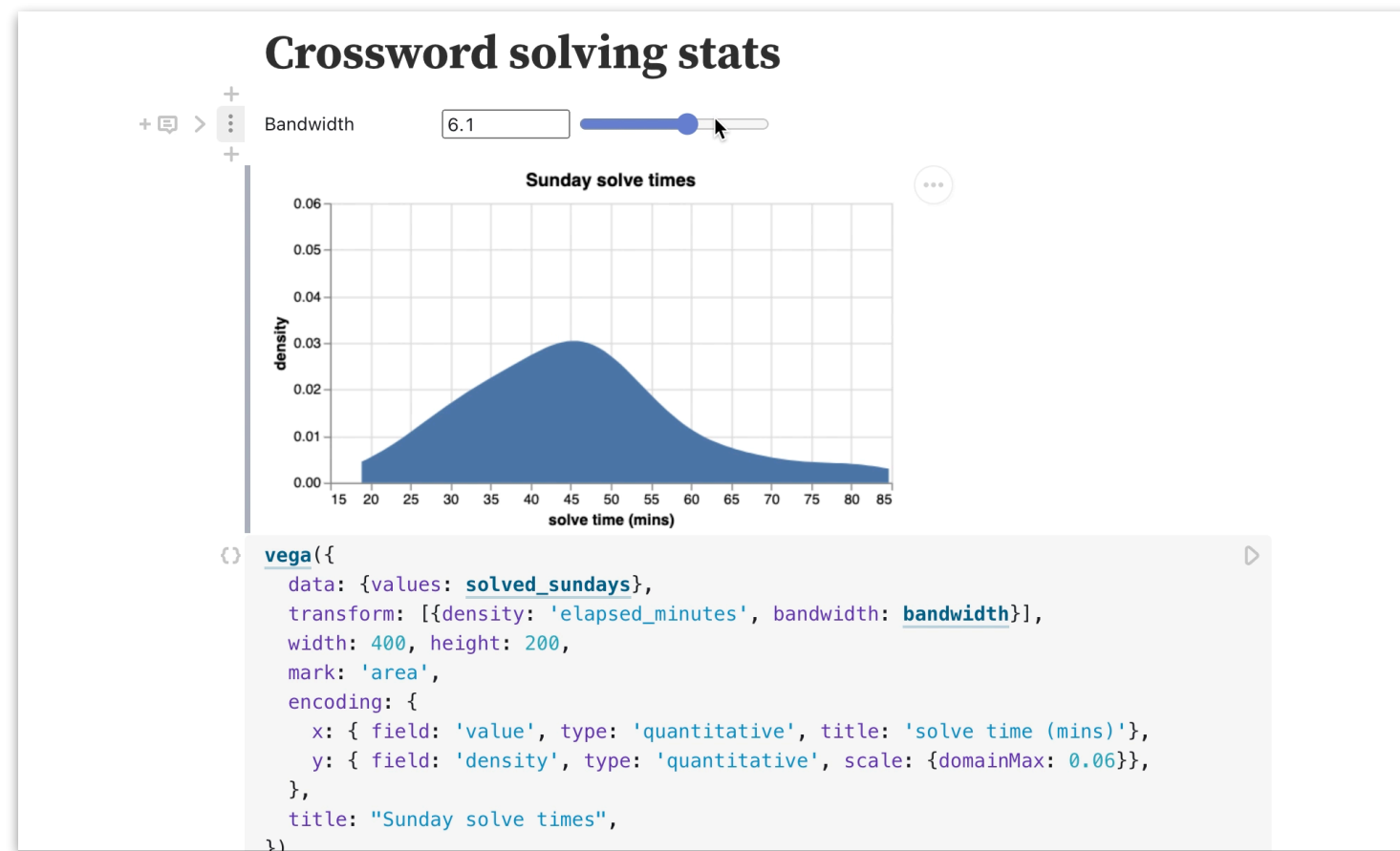


**Criticality**

# Granularity of feedback



from *Inventing on Principle* (Victor)



Observable

A screenshot of a code editor showing a Python function `average(a)` and its execution results. The function is defined as:

```
def average(a):
    s = 0
    for x in a:
        s = s + x
    return s / len(a)
```

The function is called with `average([0, 2, 5, 8, 10])`. The execution results are shown in a table:

#	a	s	x
0	[0, 2, 5, 8, 10]	0	0
1	[0, 2, 5, 8, 10]	2	2
2	[0, 2, 5, 8, 10]	7	5
3	[0, 2, 5, 8, 10]	15	8
4	[0, 2, 5, 8, 10]	25	10

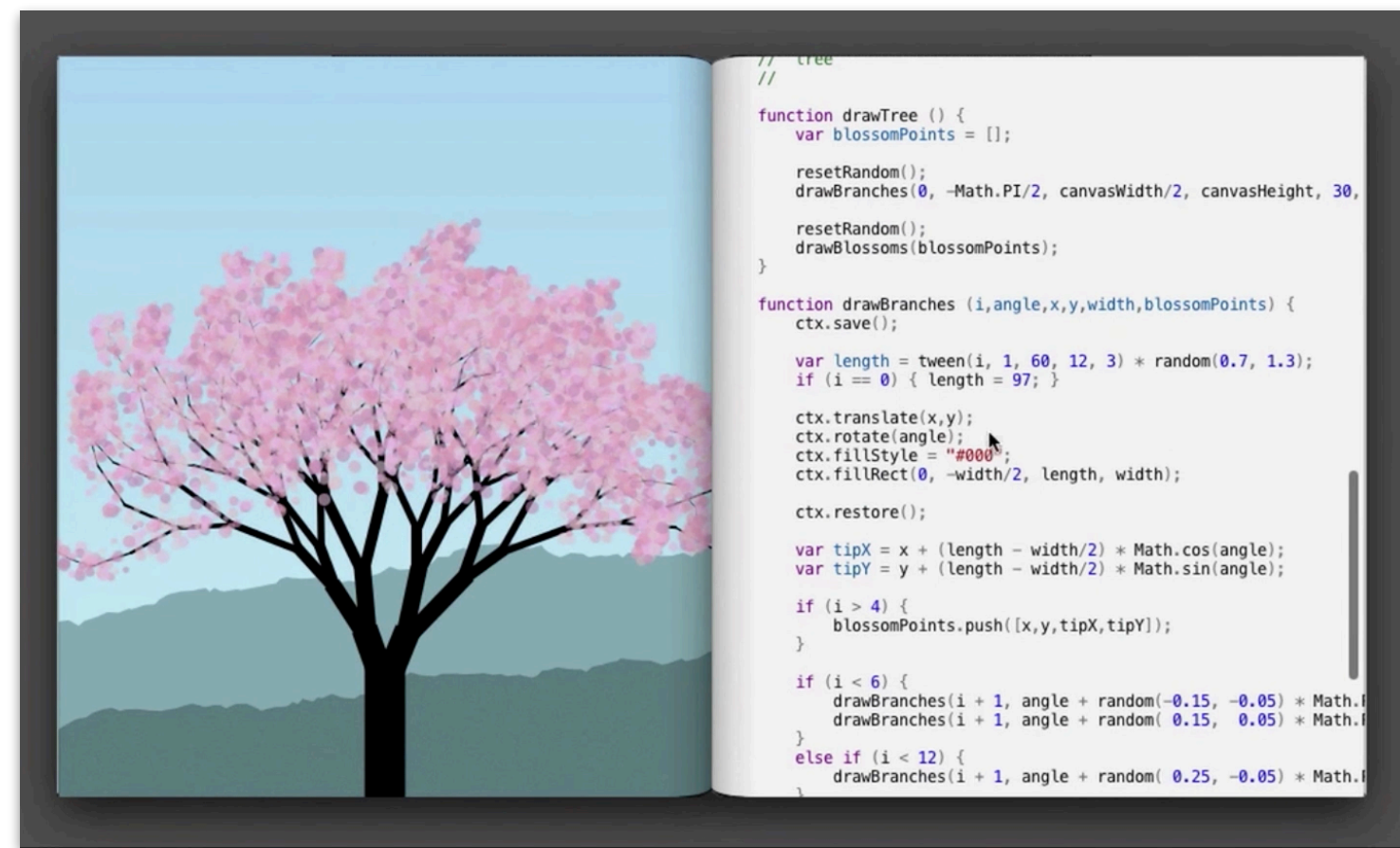
The text "2.5X speed" is displayed below the code.

Projection Boxes (Lerner)

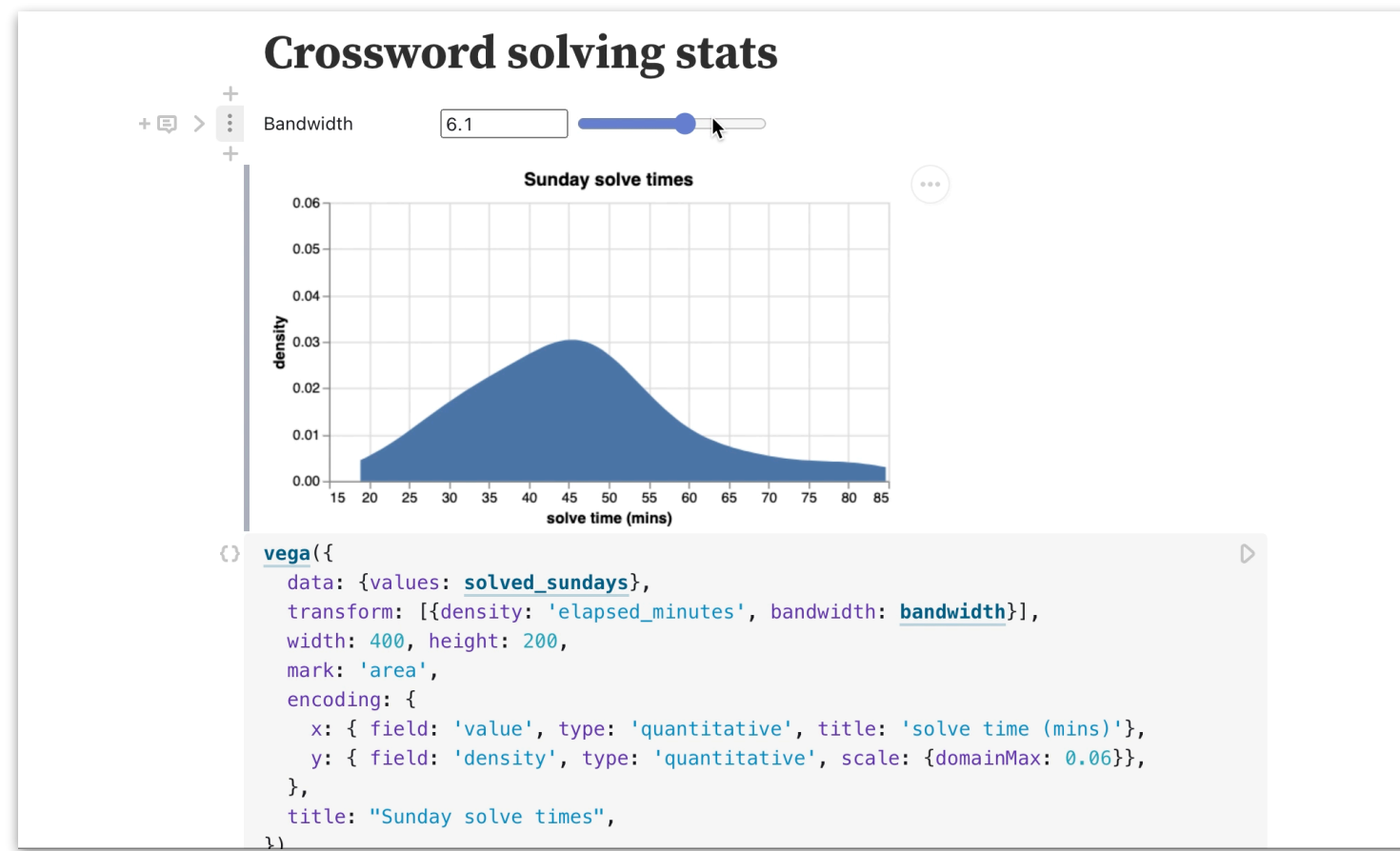


# Granularity of feedback

How deeply into the structure of a program is feedback provided?



from *Inventing on Principle* (Victor)



Observable

A screenshot of a code editor showing a Python function `average(a)`. The function calculates the average of a list of numbers. The code is as follows:

```
1
2
3 def average(a):
4     s = 0
5     for x in a:
6         s = s + x
7     l = len(a)
8     return s / l
9
10 average([0, 2, 5, 8, 10])
11
```

Three projection boxes are overlaid on the code, showing the state of variables `a`, `s`, and `x` at different points in the execution:

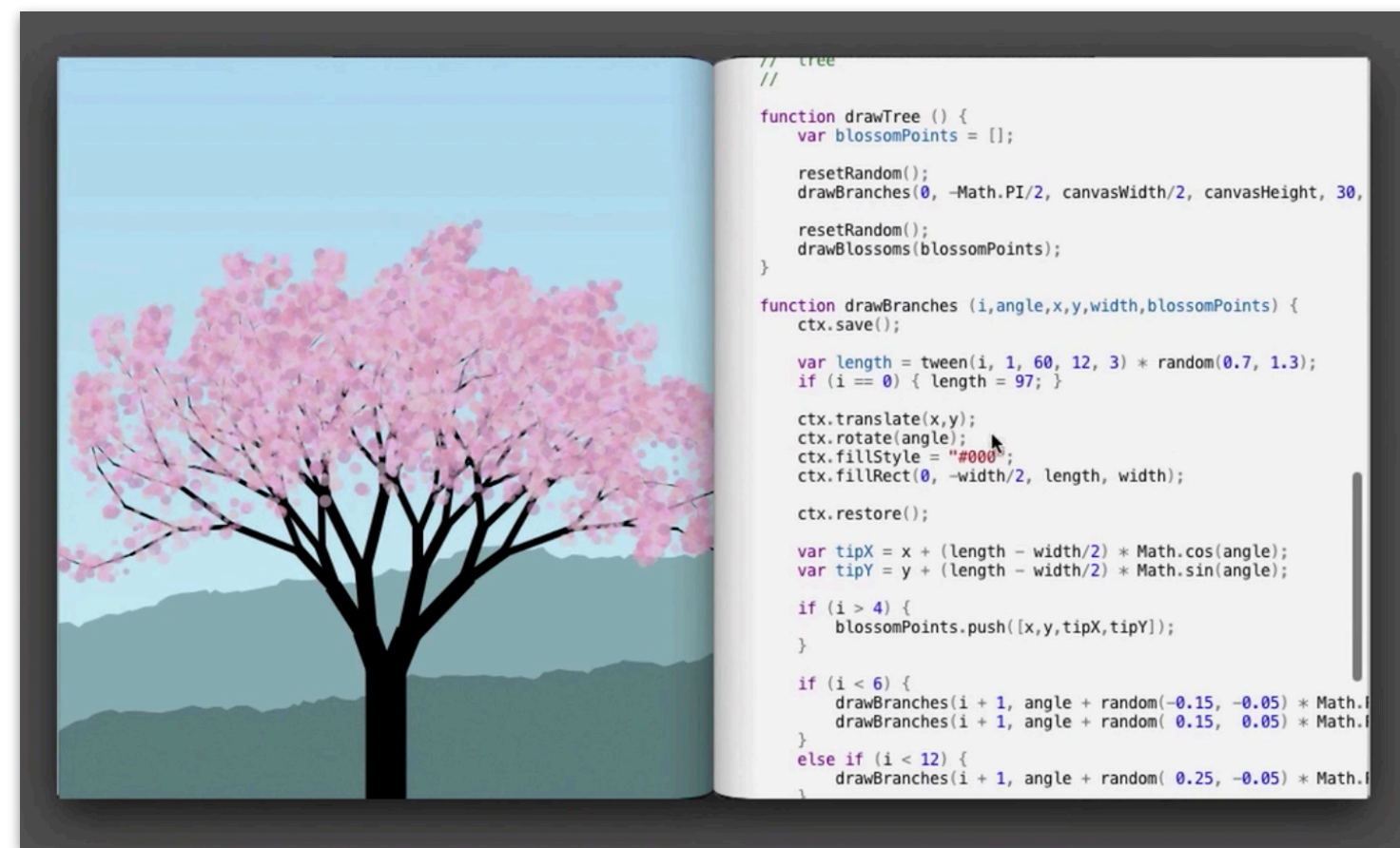
- Box 1: `a` is `[0, 2, 5, 8, 10]`, `s` is `0`, `x` is `0`.
- Box 2: `a` is `[0, 2, 5, 8, 10]`, `s` is `2`, `x` is `2`.
- Box 3: `a` is `[0, 2, 5, 8, 10]`, `s` is `7`, `x` is `5`.

The text "2.5X speed" is displayed at the bottom of the screenshot.

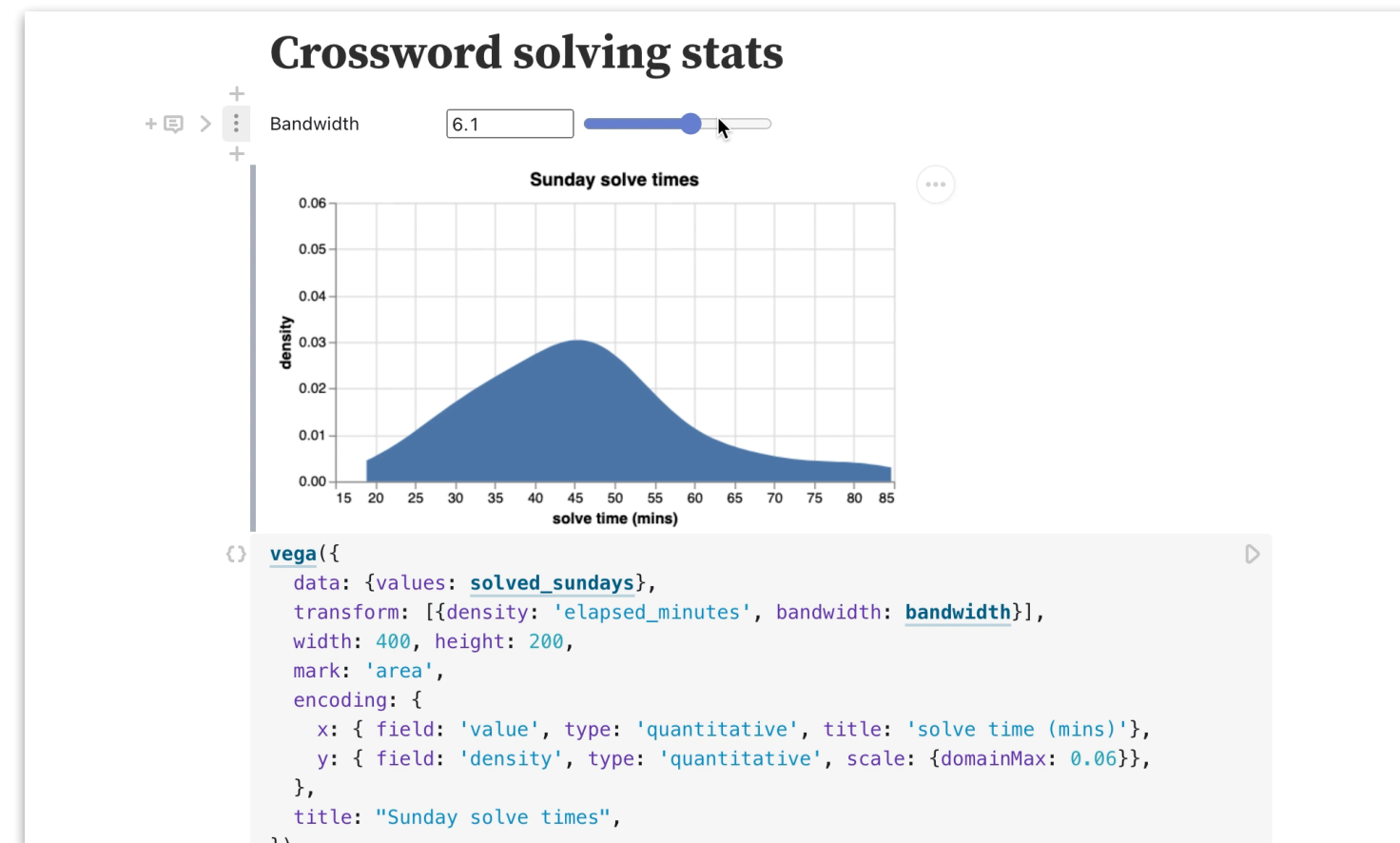
Projection Boxes (Lerner)

# Granularity of feedback

How deeply into the structure of a program is feedback provided?



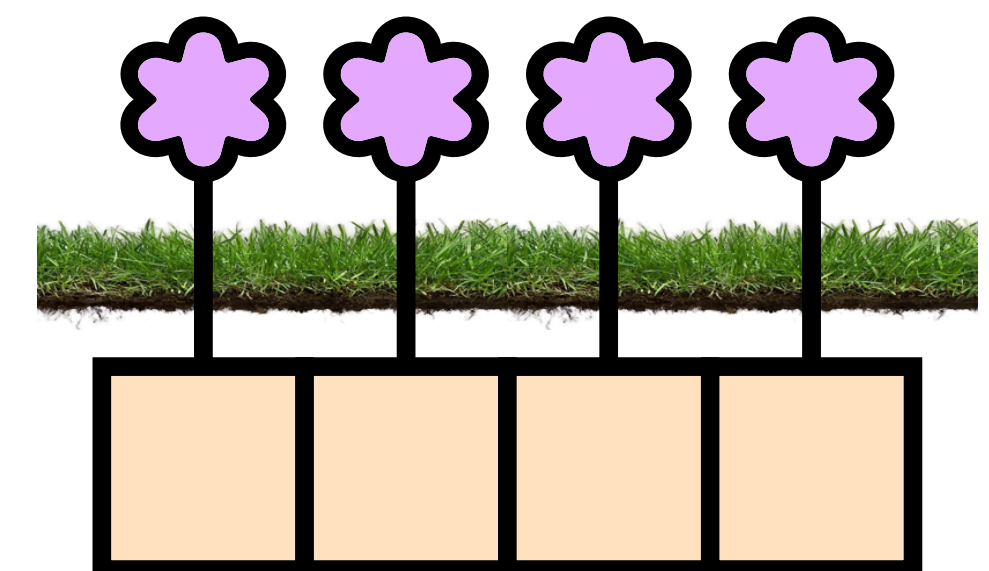
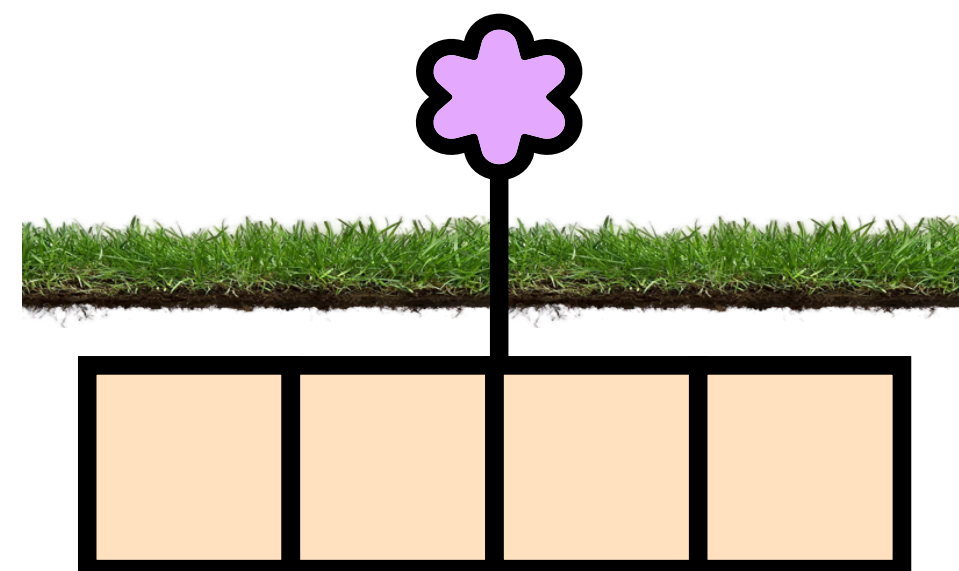
from *Inventing on Principle* (Victor)



Observable

A screenshot of a code editor with a dark background. It shows a Python function `average(a)` that iterates over an array `a` and calculates the sum `s`. Below the code, the function is called with `average([0, 2, 5, 8, 10])`. To the right, a table shows the state of variables `a`, `s`, and `x` at each step of the loop. The text "2.5X speed" is displayed at the bottom.

Projection Boxes (Lerner)

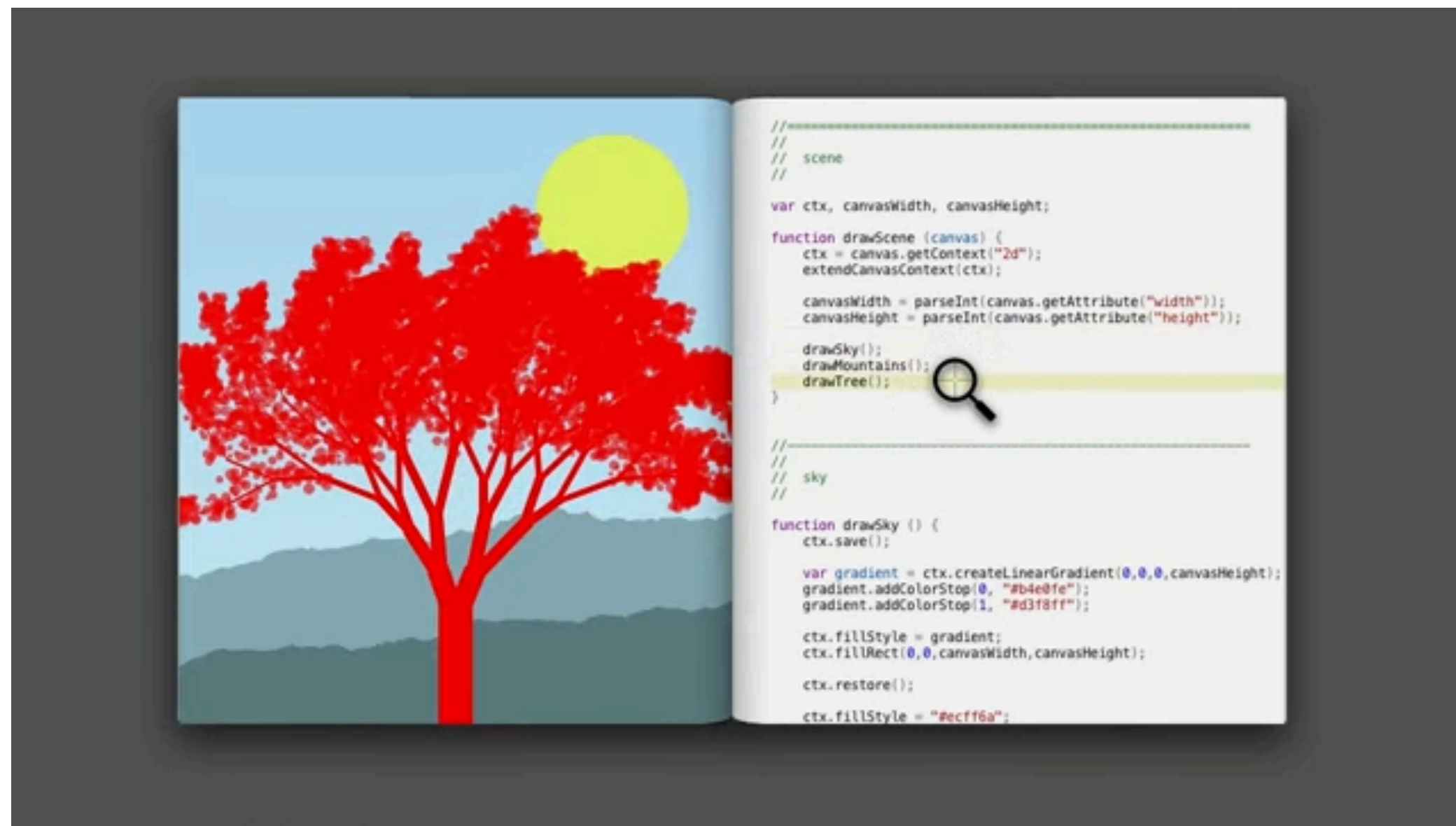


← visible output →

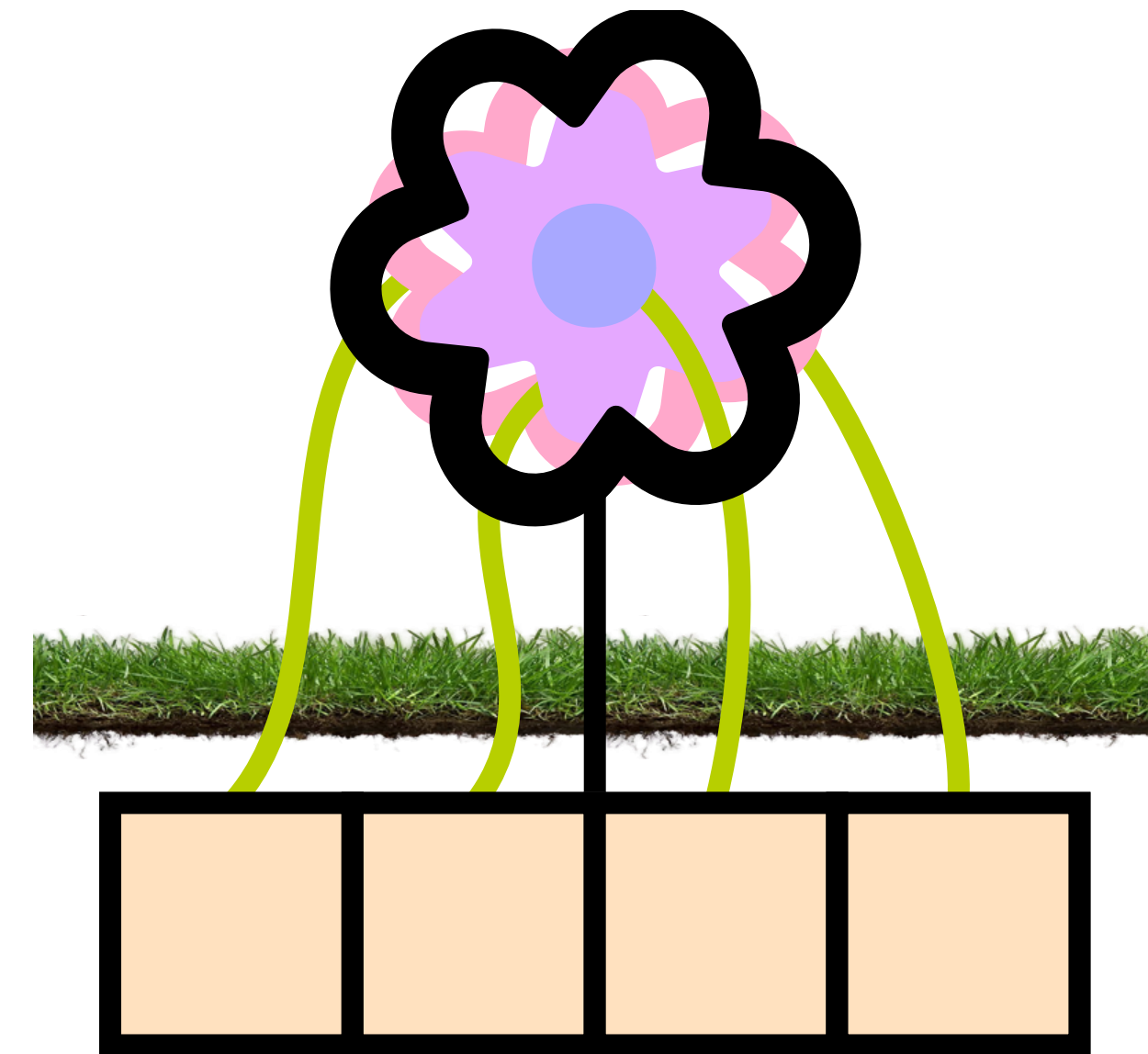
← program composed of parts →

# Granularity of feedback

How deeply into the structure of a program is feedback provided?



*still from Inventing on Principle (Victor)*

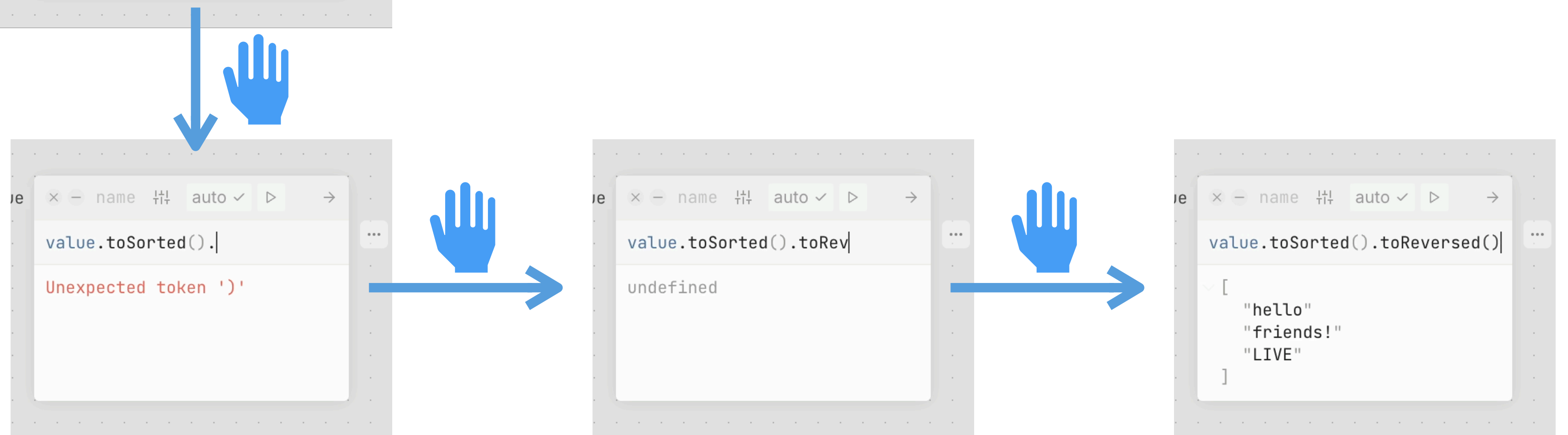


**“Linking” outputs**

# Reactivity of feedback

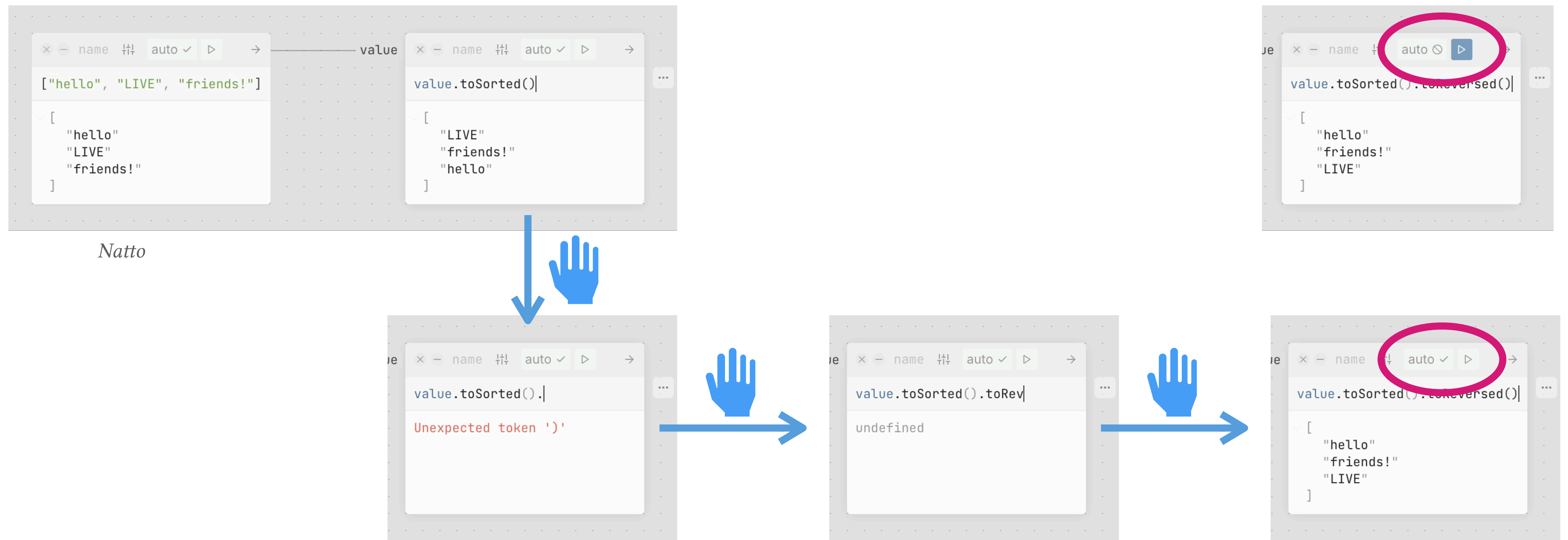
The initial state shows a variable named 'value' containing an array of strings: ["hello", "LIVE", "friends!"]. A second window shows the expression 'value.toSorted()' which has been evaluated to a new array: ["LIVE", "friends!", "hello"], demonstrating that the original array is not mutated.

*Natto*



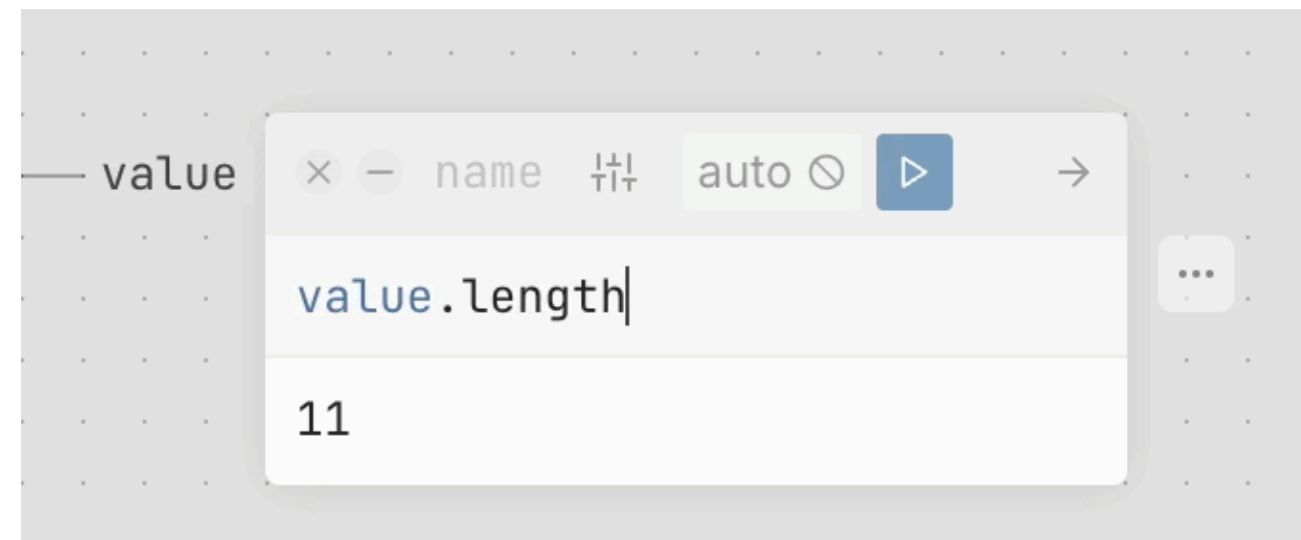
# Reactivity of feedback

How often are changes to a program responded to with feedback?

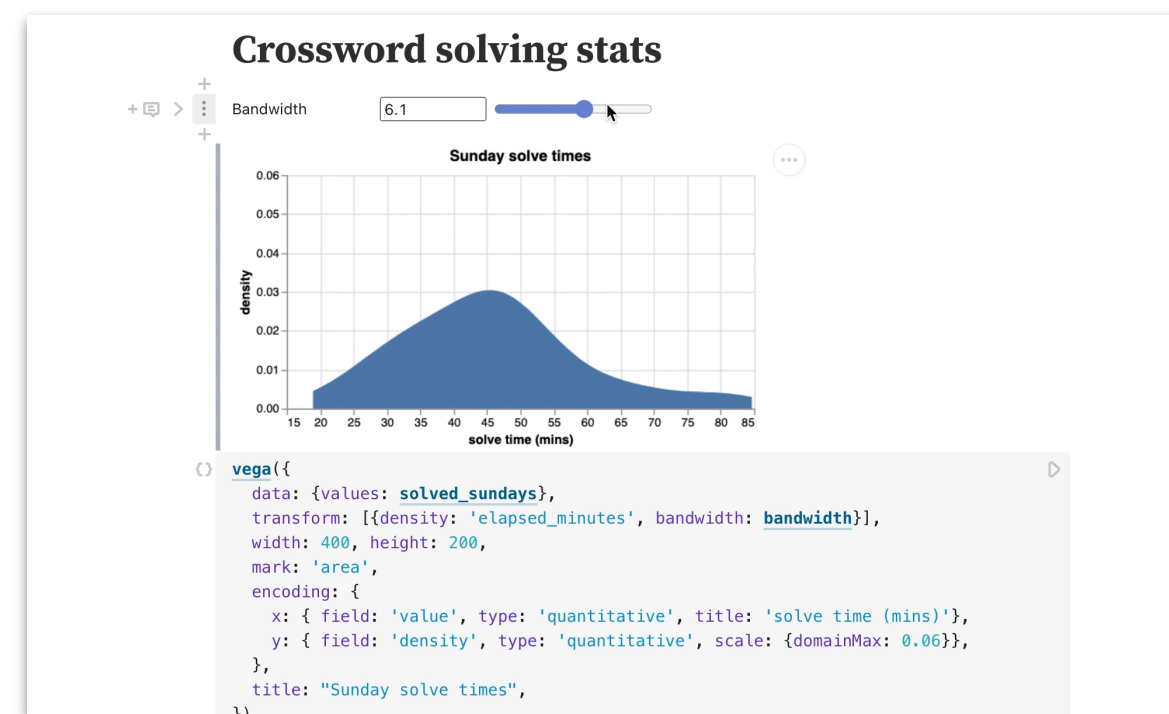


# Reactivity of feedback

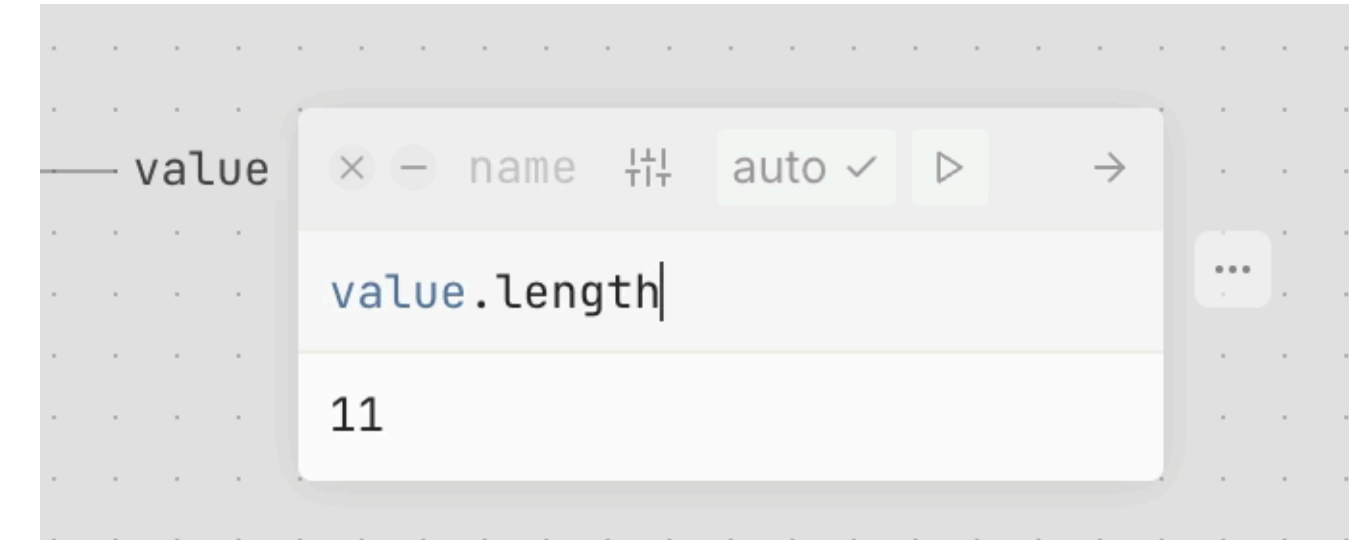
How often are changes to a program responded to with feedback?



*Observable (auto-run off)*



*Observable*

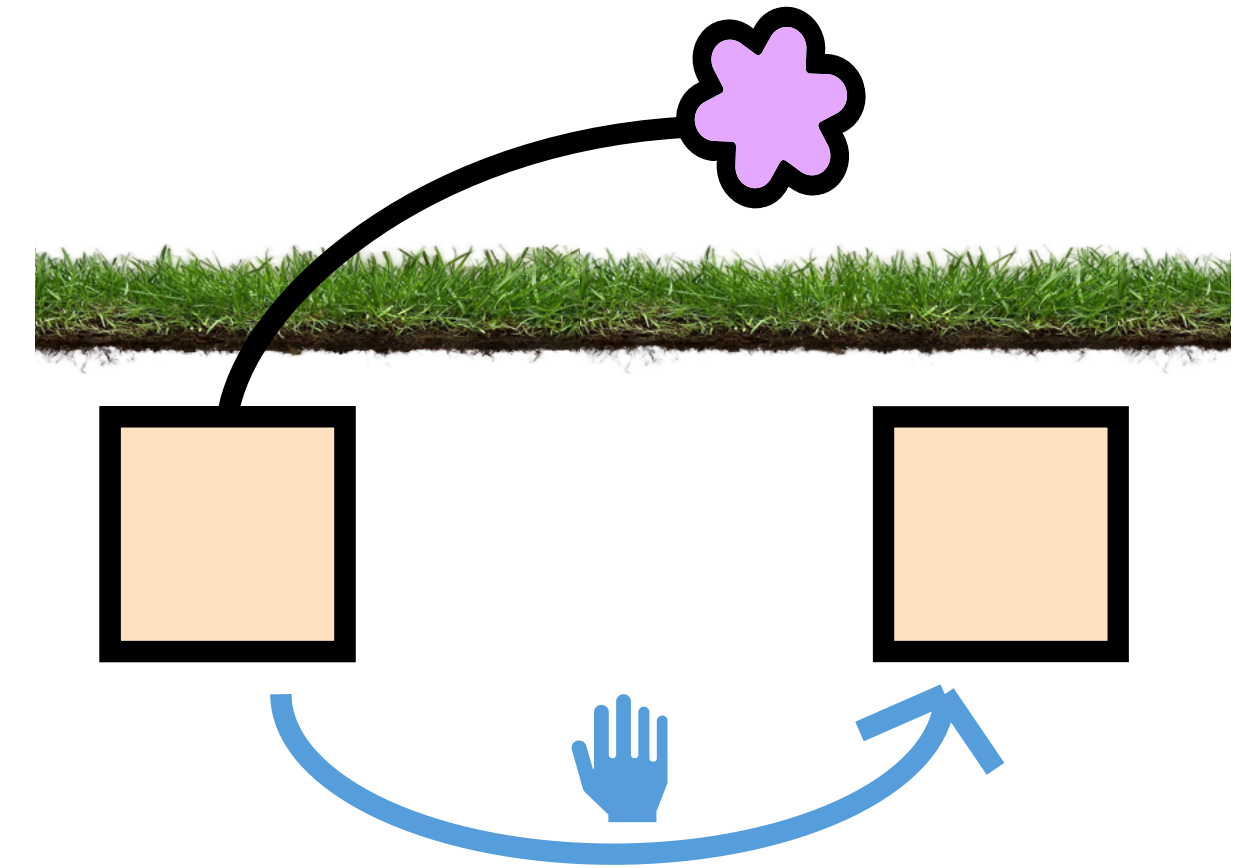
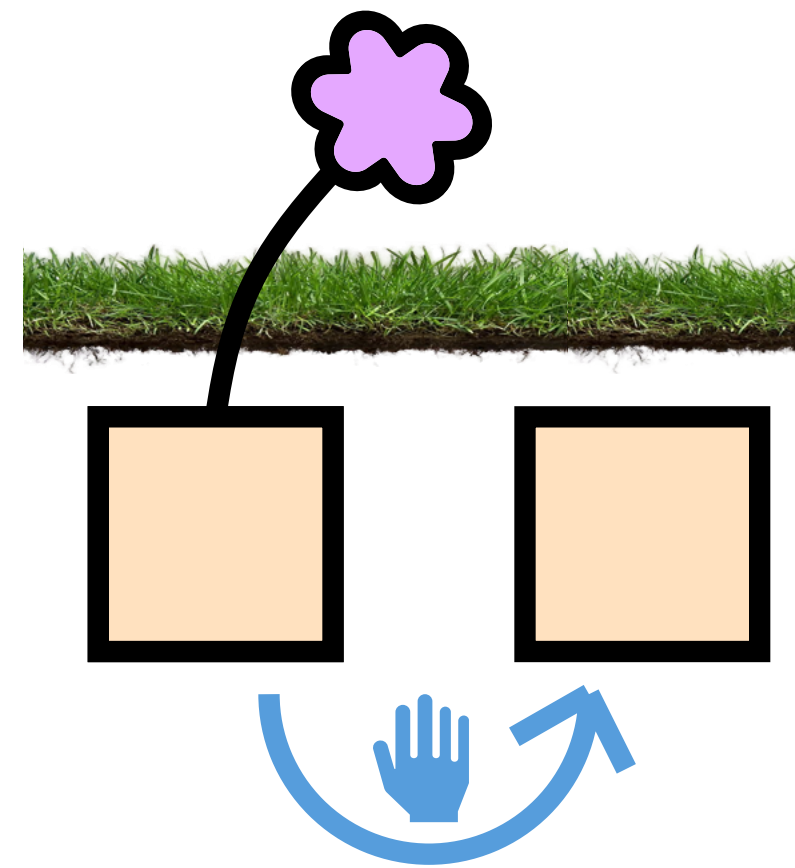
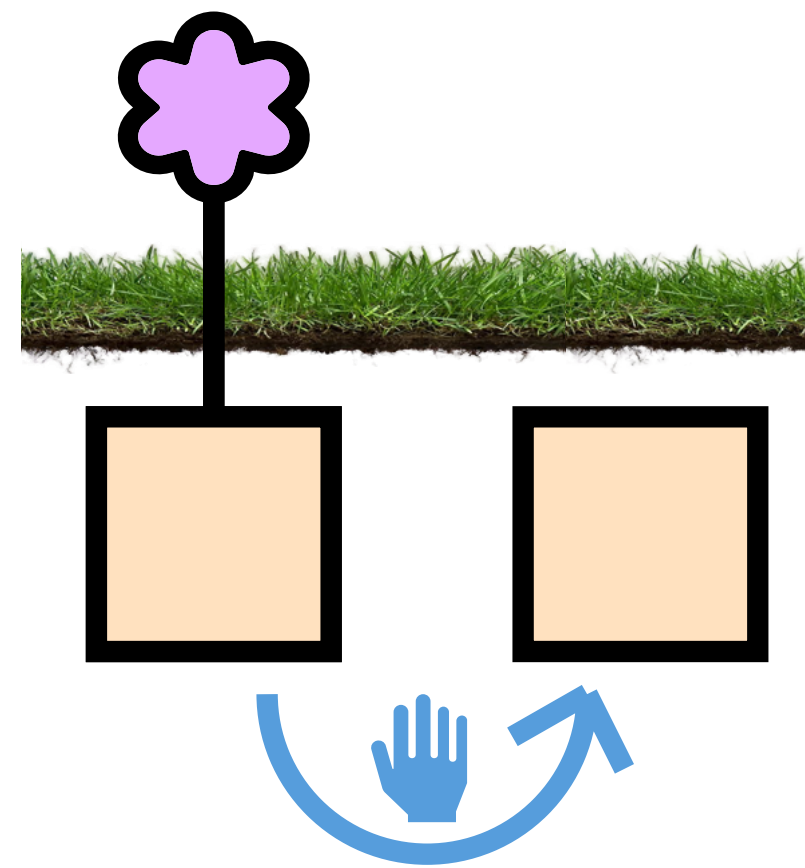


*Observable (auto-run on)*



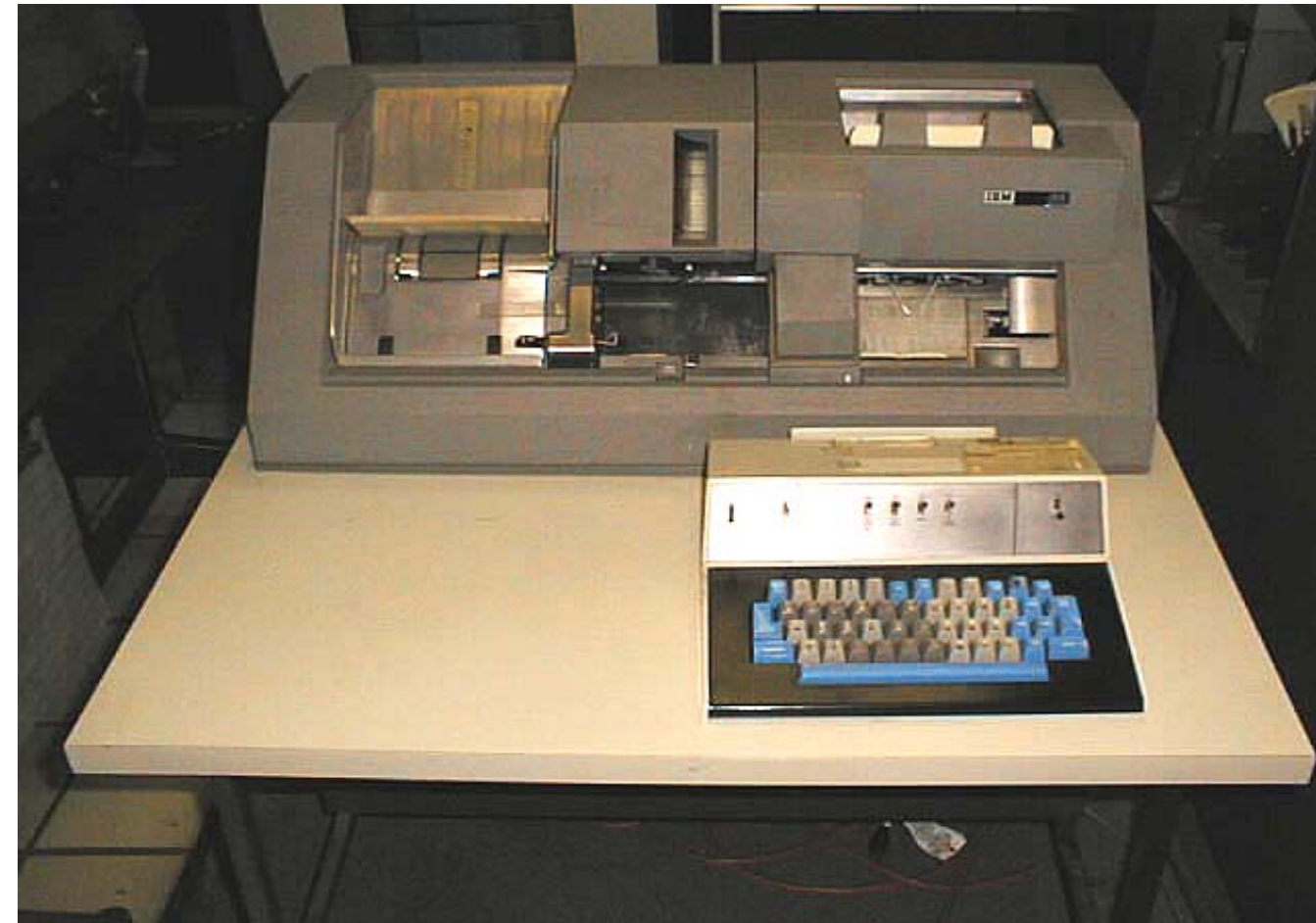
# Velocity of feedback

How quickly is feedback available?



# Velocity of feedback

How quickly is feedback available?



*IBM 029 card punch for batch computer*



*Timesharing computer*

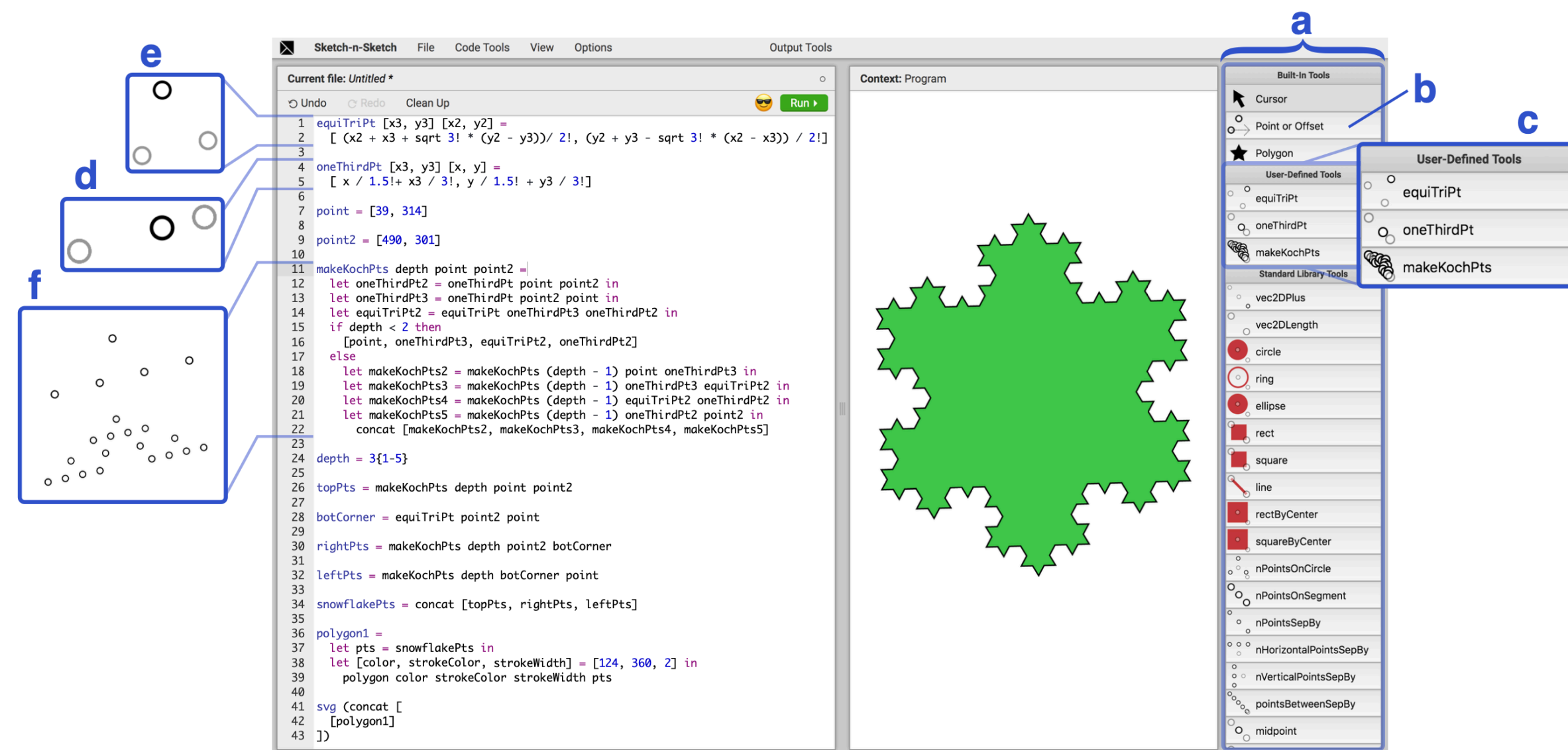
Today...

- Implementation concerns (incrementality)
- Design concerns

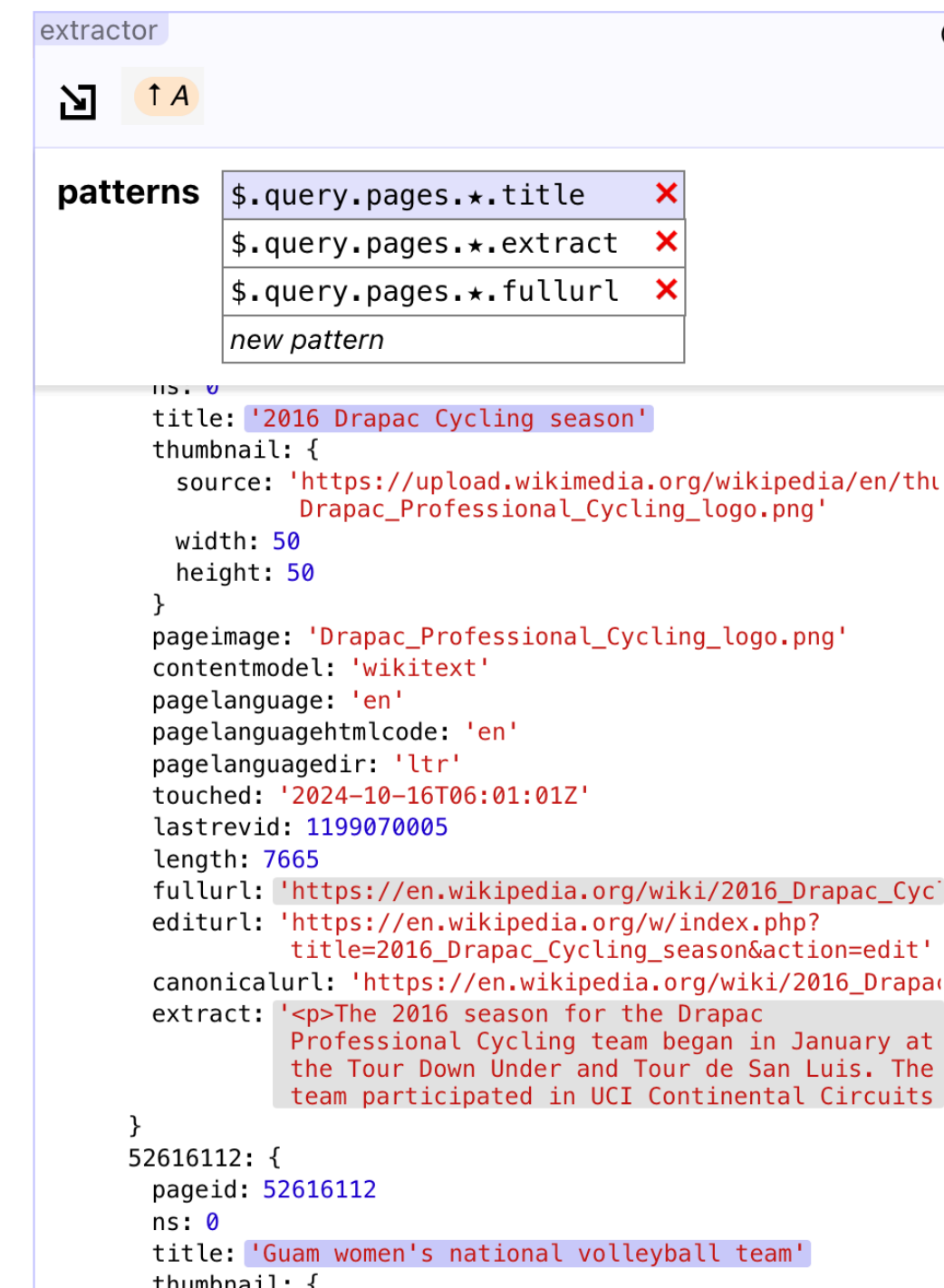


# Bidirectionality of feedback

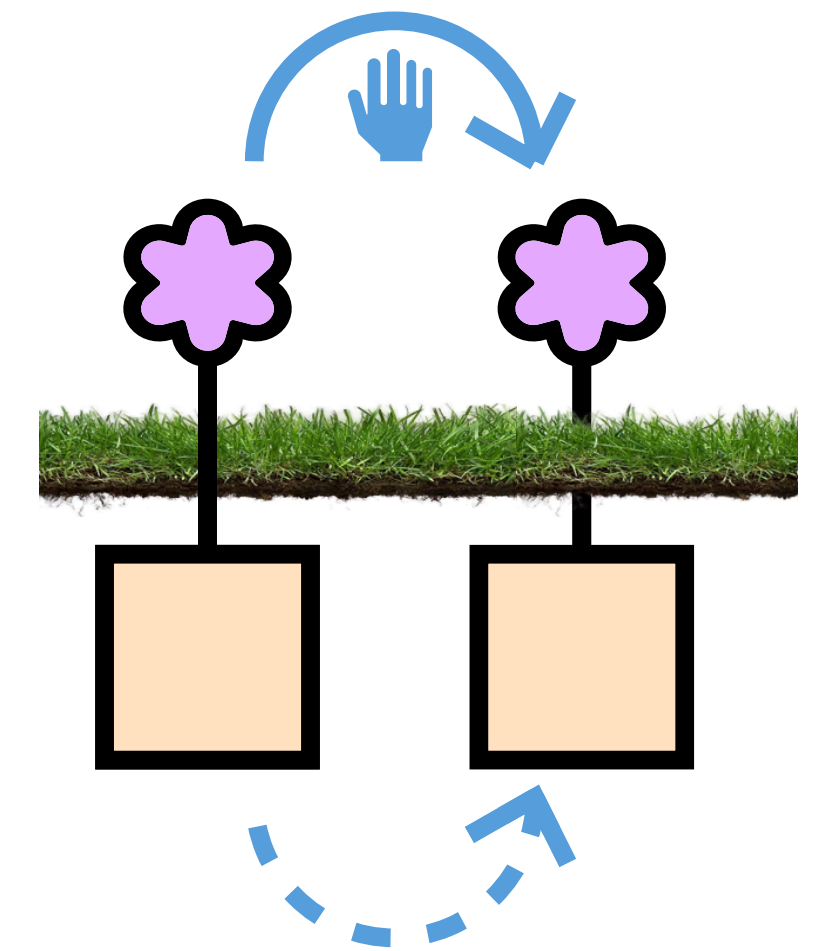
## Can programs be edited by acting on feedback?



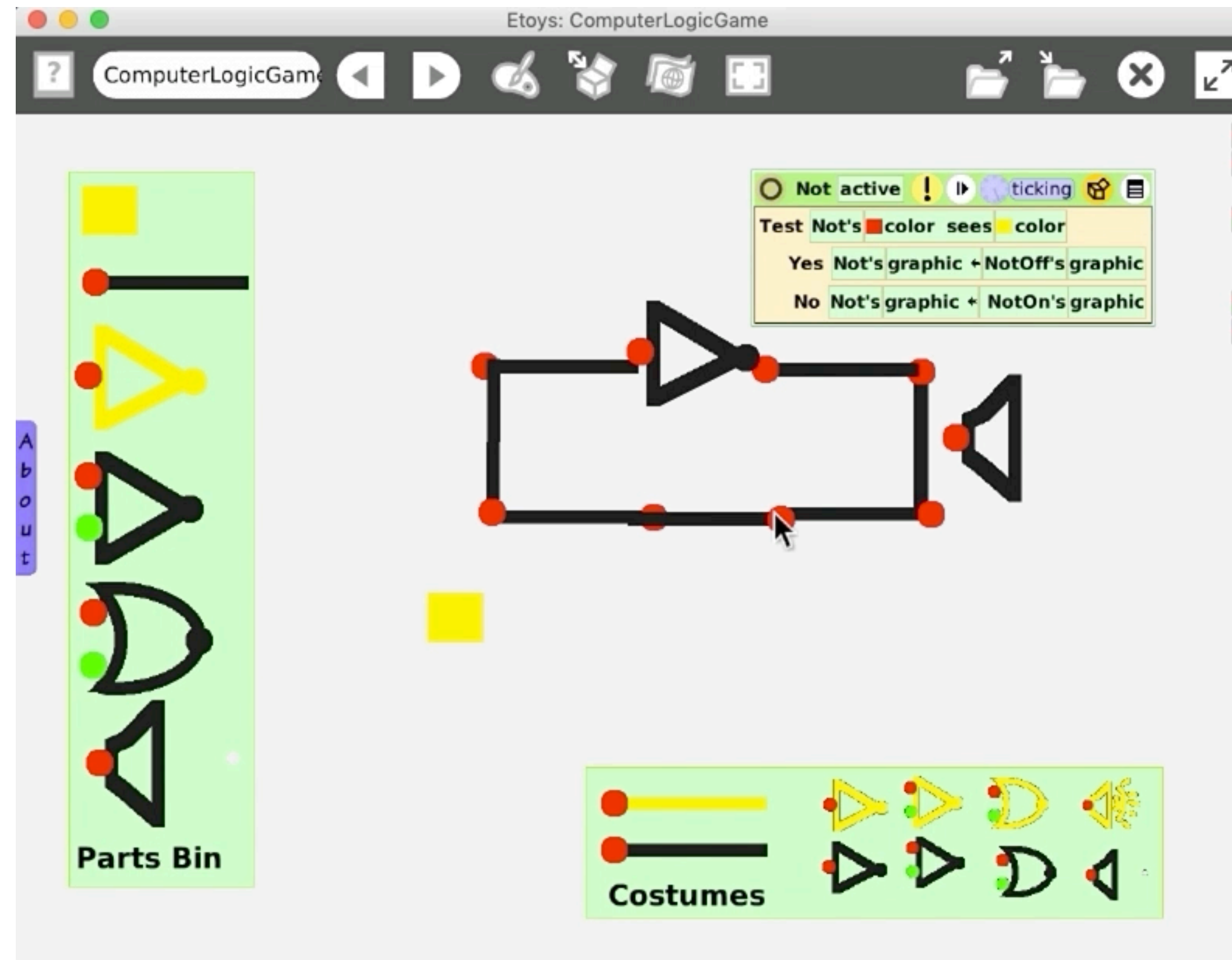
Sketch-n-Sketch (Hempel et al.)



Engraft (me)



# Criticality of feedback



# Criticality of feedback

Is feedback a side effect, or part of the critical path of computation?

```
1
2
3 def average(a):
4     s = 0
5     for x in a:
6         s = s + x
7     l =
8     return 0
9
10 average([0,2,5,8,10])
11
```

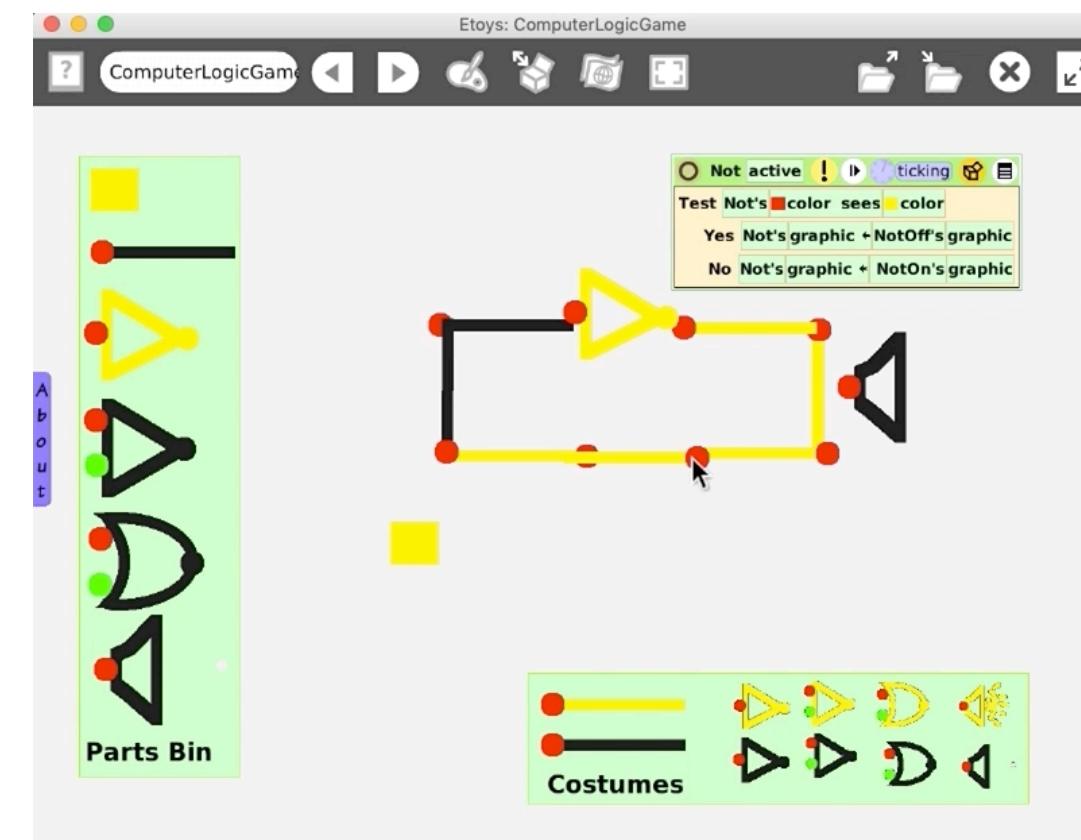
#	a	s	x
0	[0, 2, 5, 8, 10]	0	0
1	[0, 2, 5, 8, 10]	2	2
2	[0, 2, 5, 8, 10]	7	5
3	[0, 2, 5, 8, 10]	15	8
4	[0, 2, 5, 8, 10]	25	10

#	a	s	x
	[0, 2, 5, 8, 10]	25	10

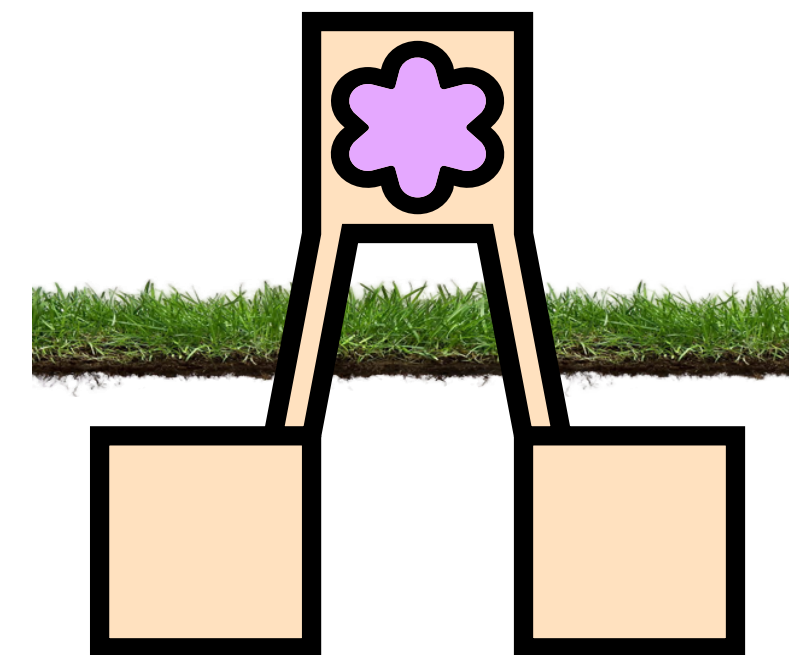
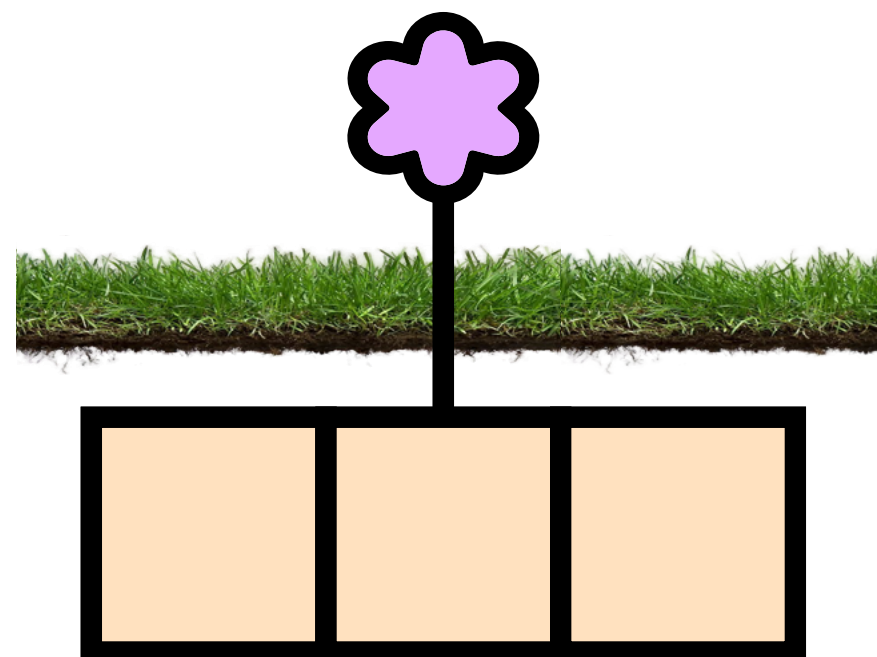
#	a	s	x	rv
	[0, 2, 5, 8, 10]	25	10	0

2.5X speed

*Projection Boxes (Lerner)*

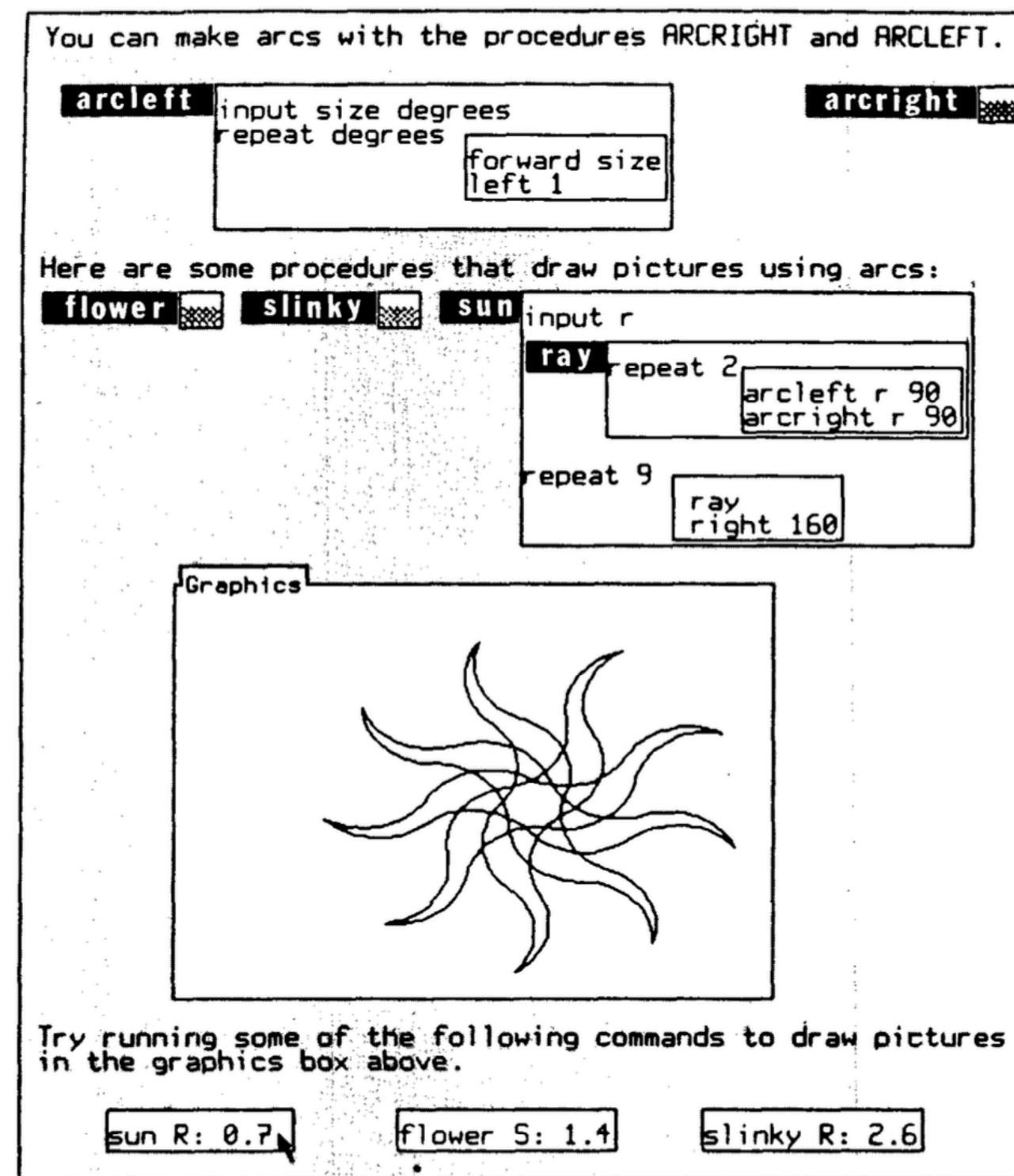


*eToys*



# Criticality of feedback

Is feedback a side effect, or part of the critical path of computation?

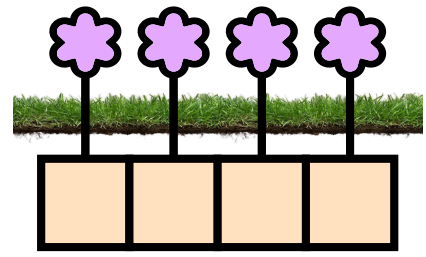


*Boxer (diSessa & Abelson)*

Naive realism is an extension of the “what you see is what you have” idea that has become commonplace in the design of text editors and spreadsheets, but not for programming languages. The point is that users should be able to pretend that what they see on the screen is their computational world in its entirety. For example, (1) any text that appears on the screen—whether typed by the system, entered by the user, or constructed by a program—can be moved, copied, modified, or (if it is program text) evaluated; (2) you can change the value of a variable simply by altering the contents of the variable box on the screen. If a program modifies the value of a variable, the contents of the box will be automatically updated on the screen. In general, there is no need to query the system to display its state, nor any need to invoke a state-change operation to affect the system indirectly.

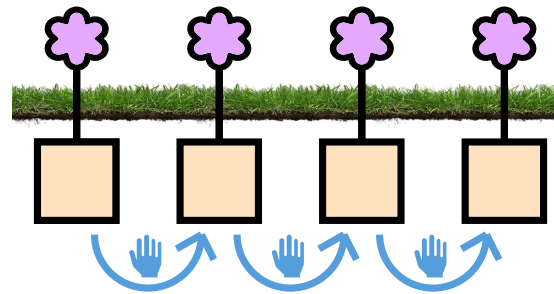
“Naive realism” in Boxer

# Technical Dimensions of *Feedback* in Live Programming Systems



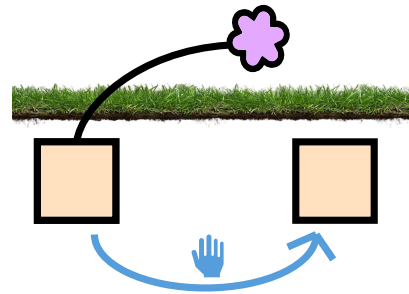
## Granularity

How deeply into the structure of a program is feedback provided?



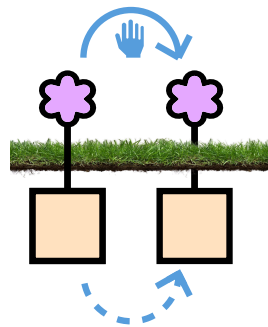
## Reactivity

How often are changes to a program reacted to with feedback?



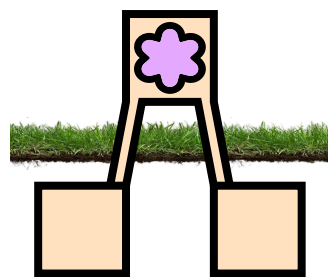
## Velocity

How quickly is feedback available?



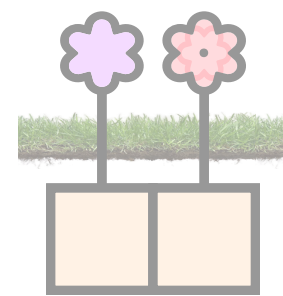
## Bidirectionality

Can programs be edited by acting on feedback?



## Criticality

Is feedback a side effect, or part of the critical path of computation?



## Moldability

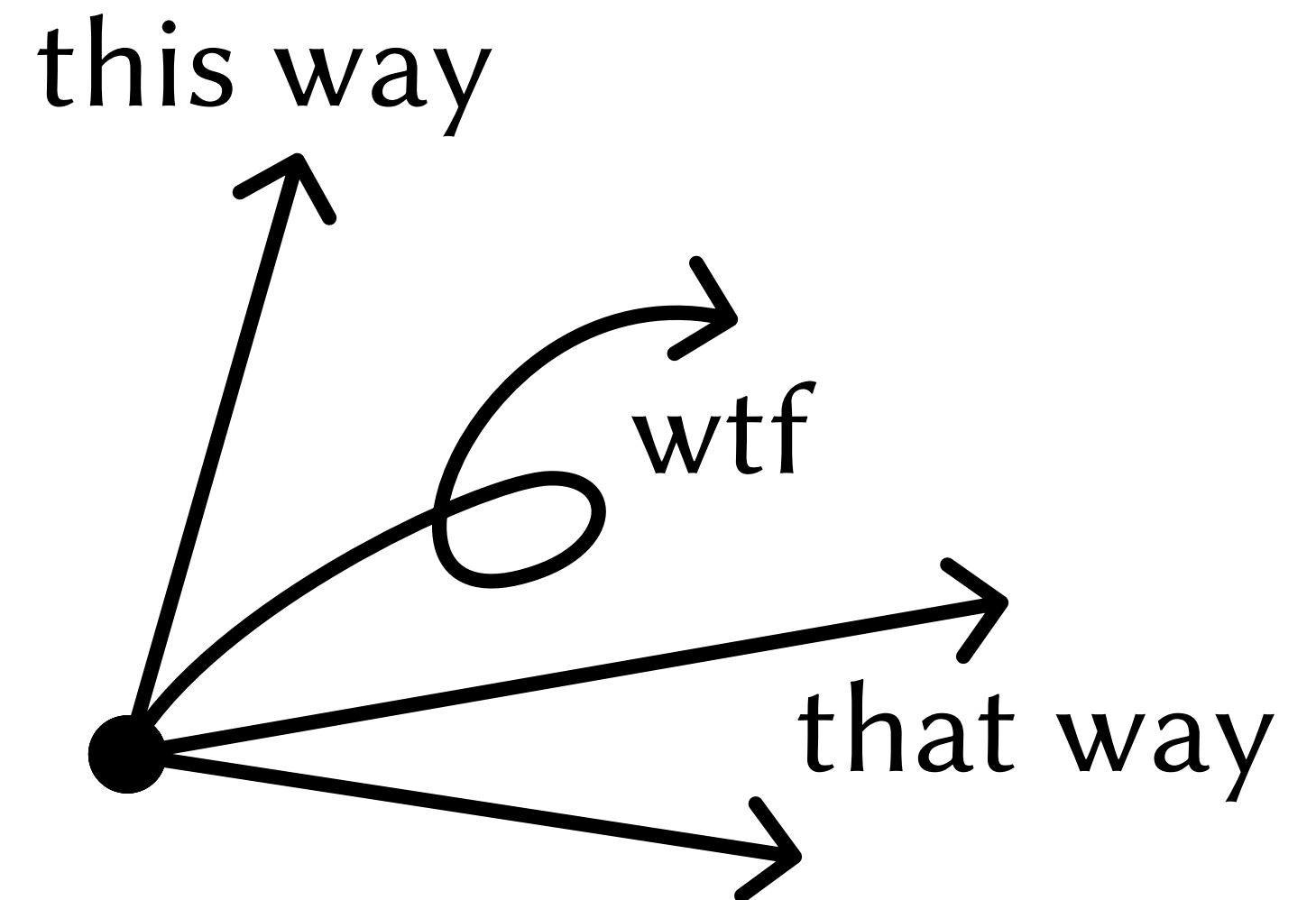
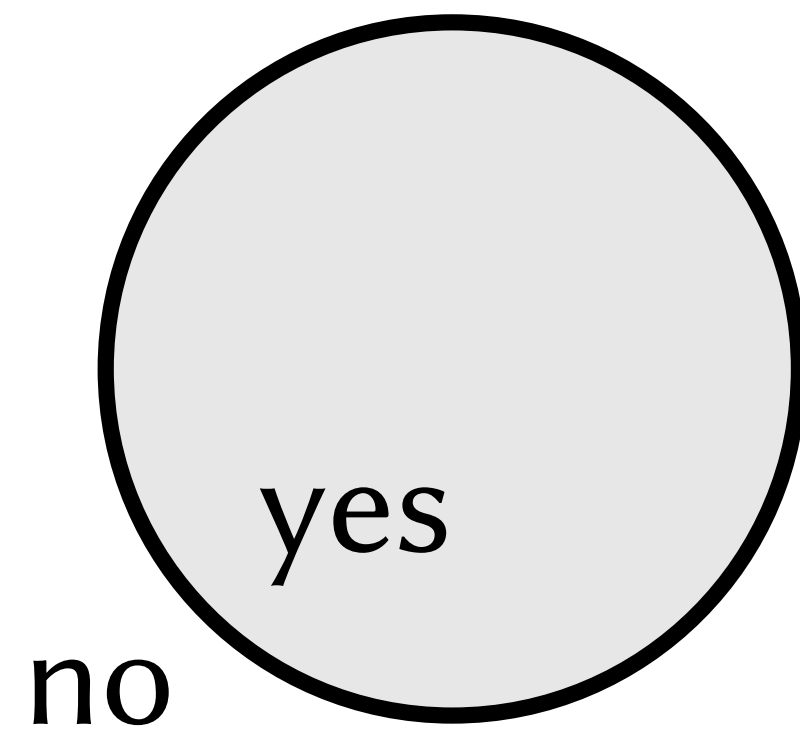
How can feedback be shaped to reflect domain-specific meaning?



# **Appendices**

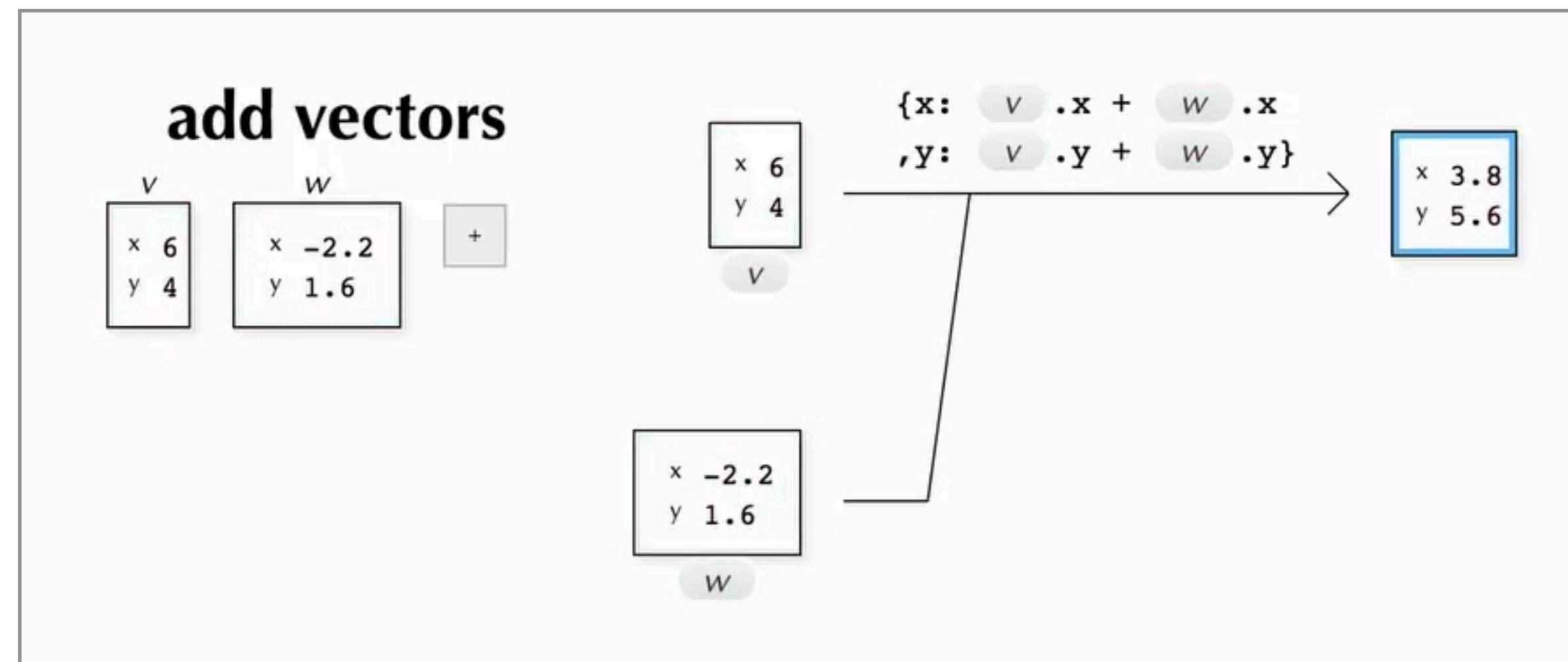
# The *Feedback Definition* of Liveness

Live programming environments provide programmers with continuous **feedback** about a program's dynamic behavior as it is being edited.



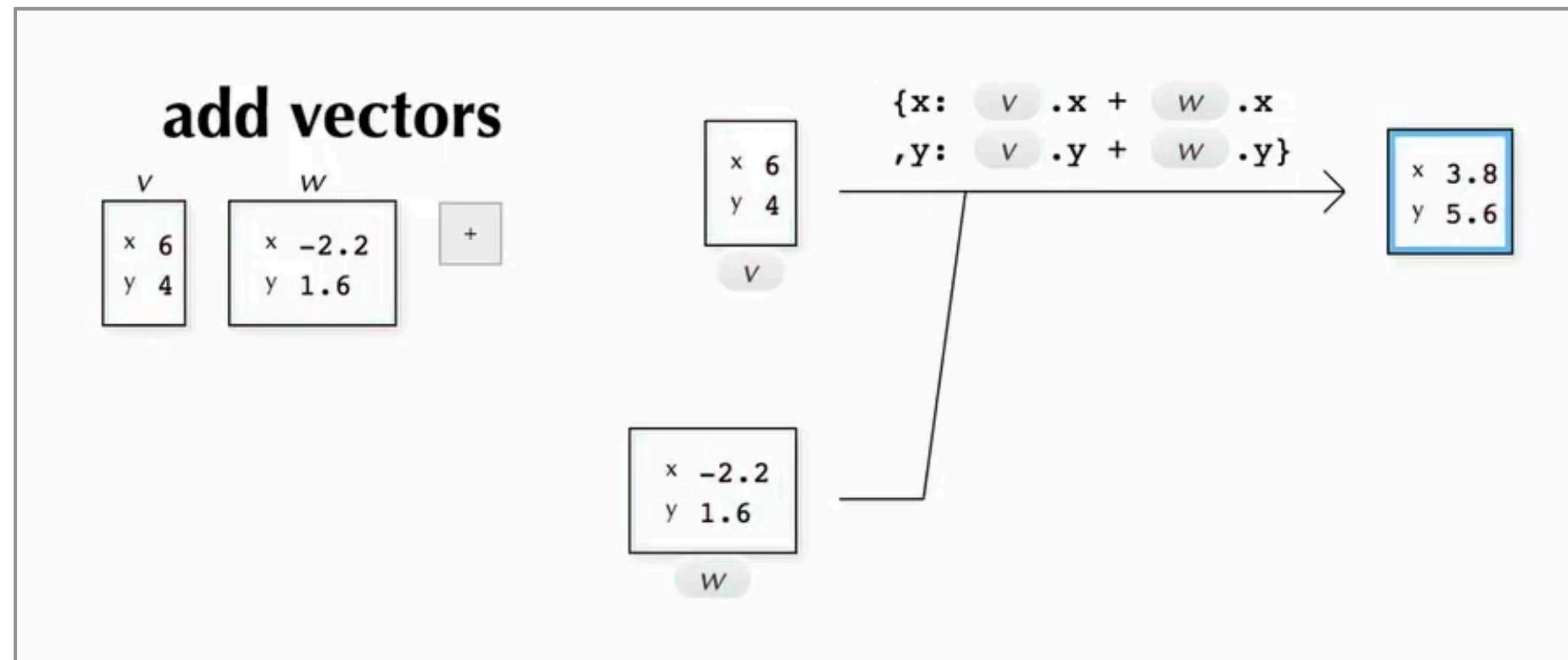


# Moldability of feedback

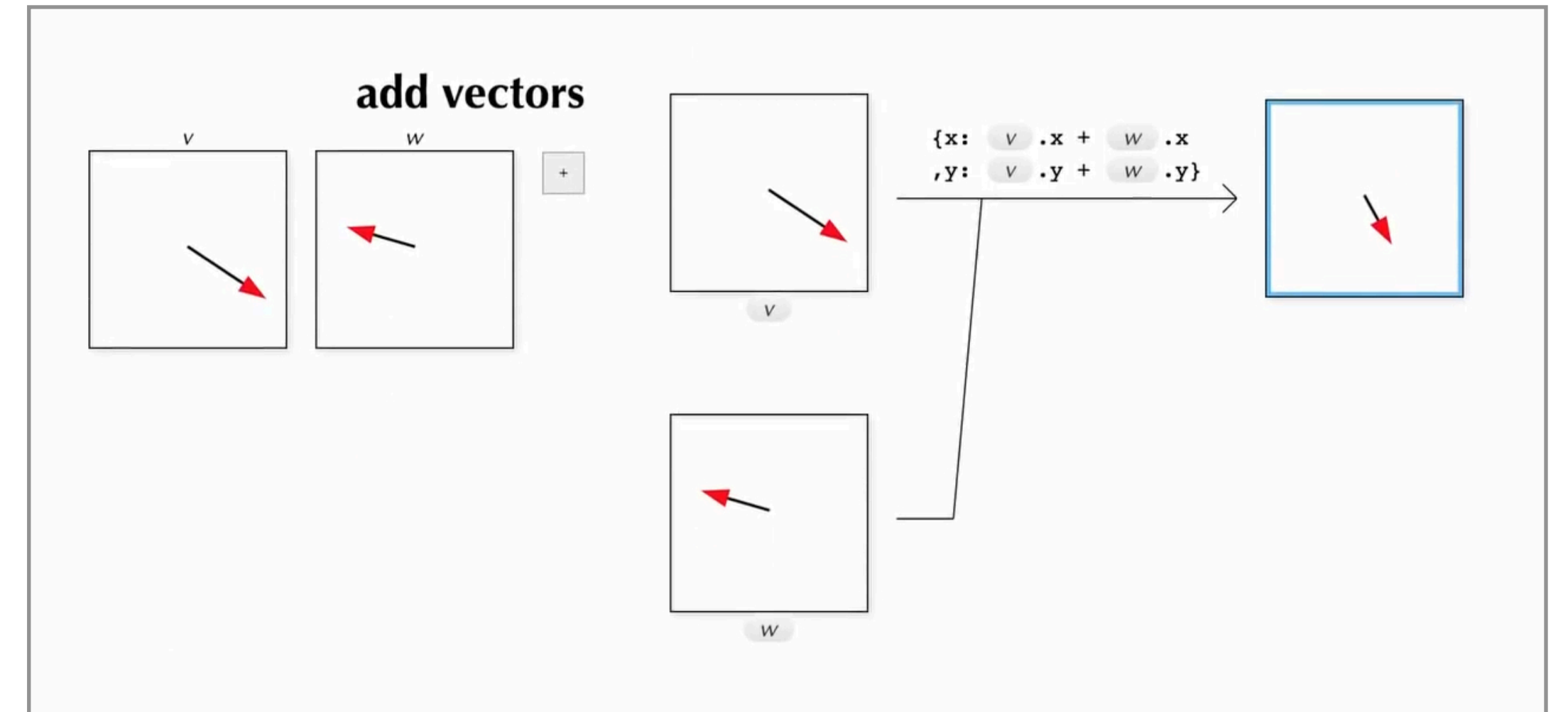


PANE (me)

# Moldability of feedback



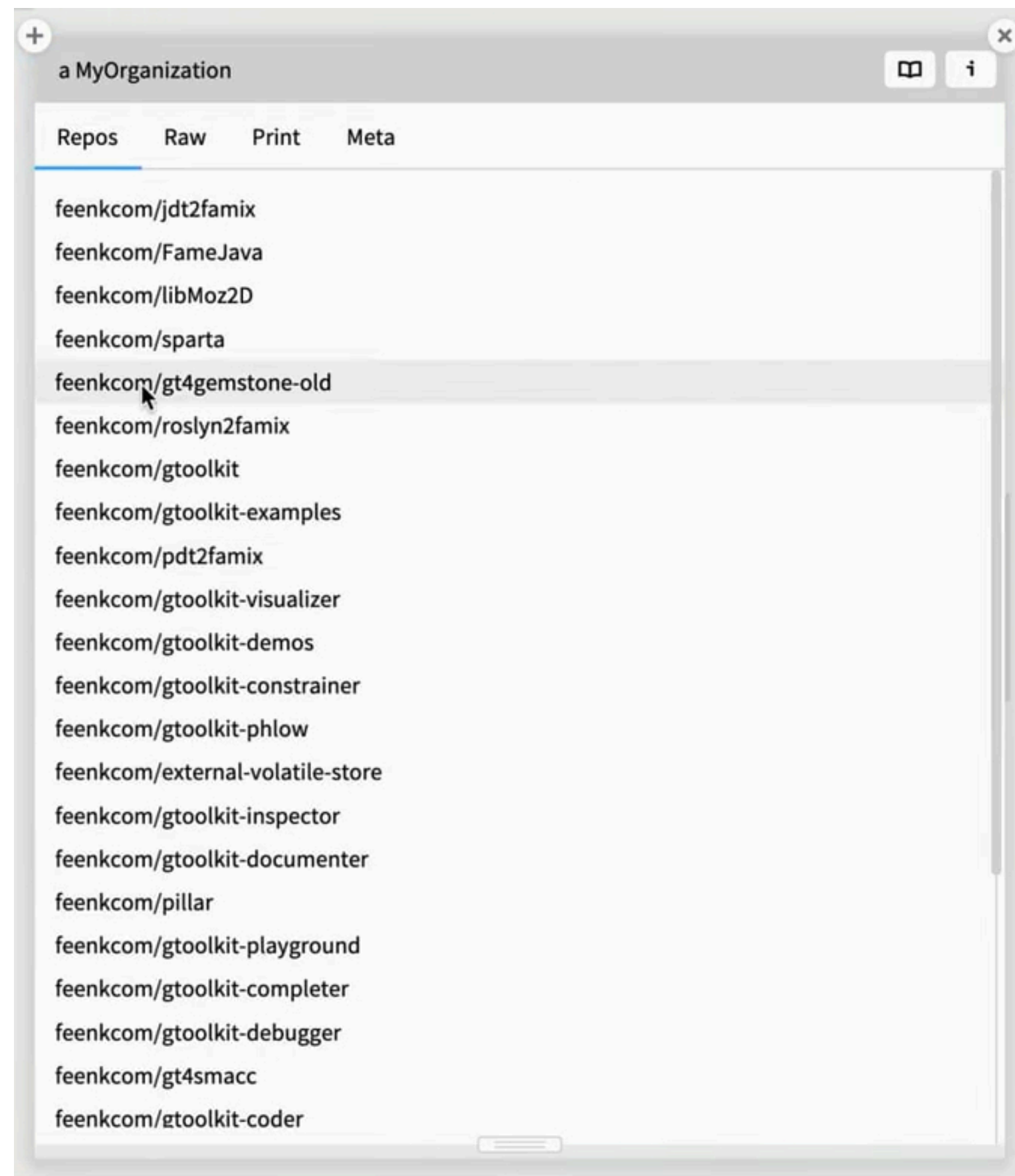
*PANE (me)*



*also PANE (still me)*

# Moldability of feedback

How can feedback be shaped to reflect domain-specific meaning?



*Glamorous Toolkit*



## Moldable Viewers

Create custom viewers for your problem at hand. Clerk's viewer API is extensible via predicate functions, not only acting on types but also on values. Build stateful viewers with Reagent and dynamically import JavaScript libraries.

*Clerk*

