Programming and Toolkits

CS 7000-001

Andrew Head & Danaé Metaxa

Announcements

Quiz 3 is next Tuesday

Starting at "Collaboration"

End at "Programming and Toolkits" (today's topic)

Final exam is during our class's finals slot during finals week, in a different room: **May 7, 2026, 12:00-2:00 PM, Towne 303**

Last time

Intelligence augmentation aims to place AI in context by using it to amplify our own abilities

Debates rage about the levels of autonomy to grant to AIs: from fully autonomous **agents** that act on the person's behalf, to **direct manipulation** that always leaves the user in full control

Mixed initiative interaction splits the difference by asking, acting, or doing nothing based on its confidence and utility

When users cannot predict how input controls affect outputs the interface, the results can be frustrating and terrible

Today

Changing problem representations

Threshold and ceiling

Learning programming

A Small Matter of Programming

Software engineering is a highly complex task, a microcosm of many challenges in HCI.

Making software engineering more accessible could empower millions to customize applications and write programs

Programming is involved

Developers **spend considerable effort recovering others' implicit knowledge** by inspecting code [LaToza, Venolia and DeLine 2006; Ko, DeLine and Venolia 2007; Ko et al. 2006]

Developers split their work between internal and external forms, e.g., **highly depending on the web** [Brandt et al. 2009]

- Just-in-time learning of new skills, clarifying existing skills
- Reminding themselves of details

Barriers span from conceptual (how is this even possible to code?) to pragmatic (how do I express this?) [Ko, Myers, and Aung 2004]

How do we aid programming?

Programming as problem representation

Cognitive amplification

By better understanding human cognition, we can design technology that makes us smarter by equipping people with the most advantageous **representations**

"The powers of cognition come from abstraction and representation: the ability to represent perceptions, experiences, and thoughts in some medium other than that in which they have occurred, abstracted away from irrelevant details." [Norman 1994]

Example: Number scrabble

[Simon 1969]

Take turns picking numbers in 1,2,3,4,5,6,7,8,9 without replacement

Win if any three of your numbers add up to 15. It's OK if you have extra numbers in your hand, as long as three of them add up to exactly 15.

An example

I will show the series of moves from players A and B so far. Raise your hand when you know what B's best next move should be.

A takes 4.
B takes 9.
A takes 2.
B takes 8.
A takes 5.

What should B do?

Re-encoding number scrabble

4	9	2
3	5	7
8	1	6

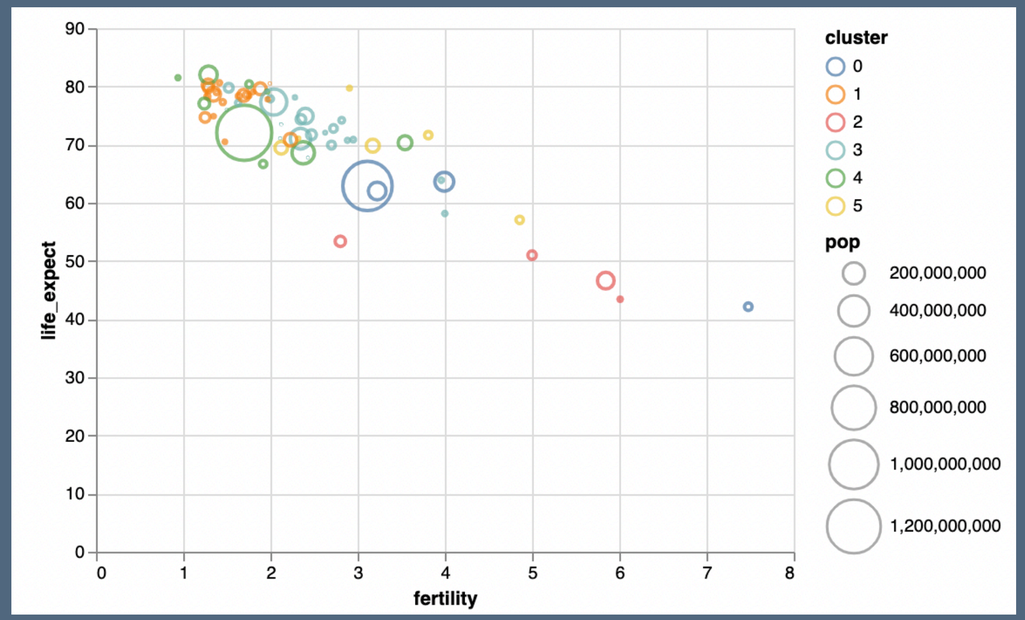
Ready, set, go!

A	B	A
	A	
B		

Representations for vis

[Bertin 1983; Mackinlay 1986; Satyanarayan 2016]

How do we tell a machine to create this? Paint pixels?



It's extremely challenging until we adopt a representation that visualizations are **encodings of data types into marks**

```
vl.markPoint()
  .data(data2000)
  .encode(
    vl.x().fieldQ('fertility'),
    vl.y().fieldQ('life_expect'),
    vl.size().fieldQ('pop').scale({range: [0, 1000]}),
    vl.color().fieldN('cluster')
  )
  .render()
```

Domain-specific languages

DSLs, or domain-specific languages, are programming languages that are tailored to a specific domain

SQL (databases)

d3 / Vega Lite (visualization)

pytorch, keras, tensor flow (machine learning)

Successful DSLs **reshape the cognitive representation** of the task, reducing the gulfs of execution and evaluation and empowering development in their application domain

Data science representations

I have too much data to fit in my computer. How do I count the number of times the word "HCI" appears on the web?

Representation: ***Map-Reduce*** [Dean and Ghemawat 2008]

> First, run a ***Map*** phase that runs a simple function over each webpage. That function outputs the number of HCIs, and can be run completely in parallel across every page on the web
>
> Second, run a ***Reduce*** phase that collects the outputs from the Map phase and aggregates them: here, via a sum

Threshold and Ceiling

What is your programming intervention actually doing?

What is Github Copilot's design goal? How do we know if it's succeeding at that design goal?

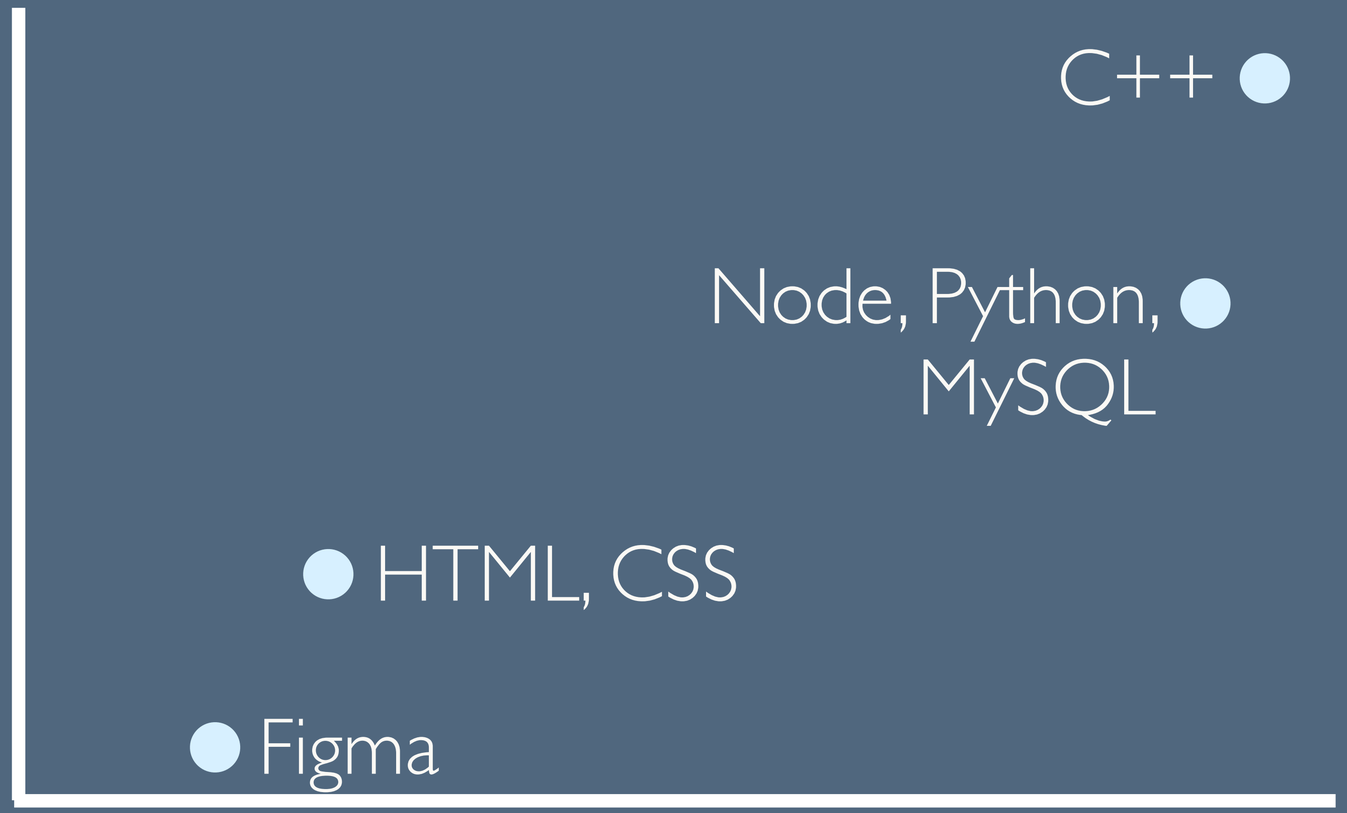
Are some programming languages "better" than others? How would we know?

Is the VSCode plugin helping? With what?

Threshold/Ceiling Diagram

[Myers, Hudson and Pausch, TOCHI 2000]

Threshold: Difficulty of non-expert use



Are you trying to **lower the threshold**, or **raise the ceiling**?

Ceiling: Expressivity; sophistication of what can be created

Tradeoffs

Threshold is about **ease of use for non-experts**;
Ceiling is about **expressivity for experts**

Often, threshold and ceiling are in tension with each other.

The **command line** and **Photoshop** are **high ceiling, but also high threshold** because they require substantial up-front learning

A tool's threshold or ceiling status **may depend on the user's background**: you might be able to make striking representational art with a **simple digital brush**, even if I can only produce stuff that looks like bad graffiti

Some tools enable you to **"pop out" to code or advanced tools to achieve higher ceilings** for experts: e.g., Microsoft Word has a low threshold, but you can do a lot of complex layout with it if you know how

Lowering the threshold

Goal: reduce the effort and cognitive complexity of creating software artifacts

Tools for statistical analysis

[Jun et al. 2019]

Non-experts often struggle to **select appropriate statistical tests**

So, instead of asking people to directly run statistical tests, instead ask them to write down the **properties of their method and data**

This representation **lowers the threshold to statistical test selection**, but **limits the ceiling** of some complex models

```
import tea
tea.data('UScrime.csv')
variables = [
    {
        'name' : 'So',
        'data type' : 'nominal',
        'categories' : ['0', '1']
    },
    {
        'name' : 'Prob',
        'data type' : 'ratio',
        'range' : [0,1]
    }
]
tea.define_variables(variables)
study_design = {
                    'study type': 'observational study',
                    'contributor variables': 'So',
                    'outcome variables': 'Prob',
                }
tea.define_study_design(study_design)
assumptions = {
    'groups normally distributed': [['So', 'Prob']],
    'Type I (False Positive) Error Rate': 0.05
}
tea.assume(assumptions)
hypothesis = 'So:1 > 0'
tea.hypothesize(['So', 'Prob'], hypothesis)
```

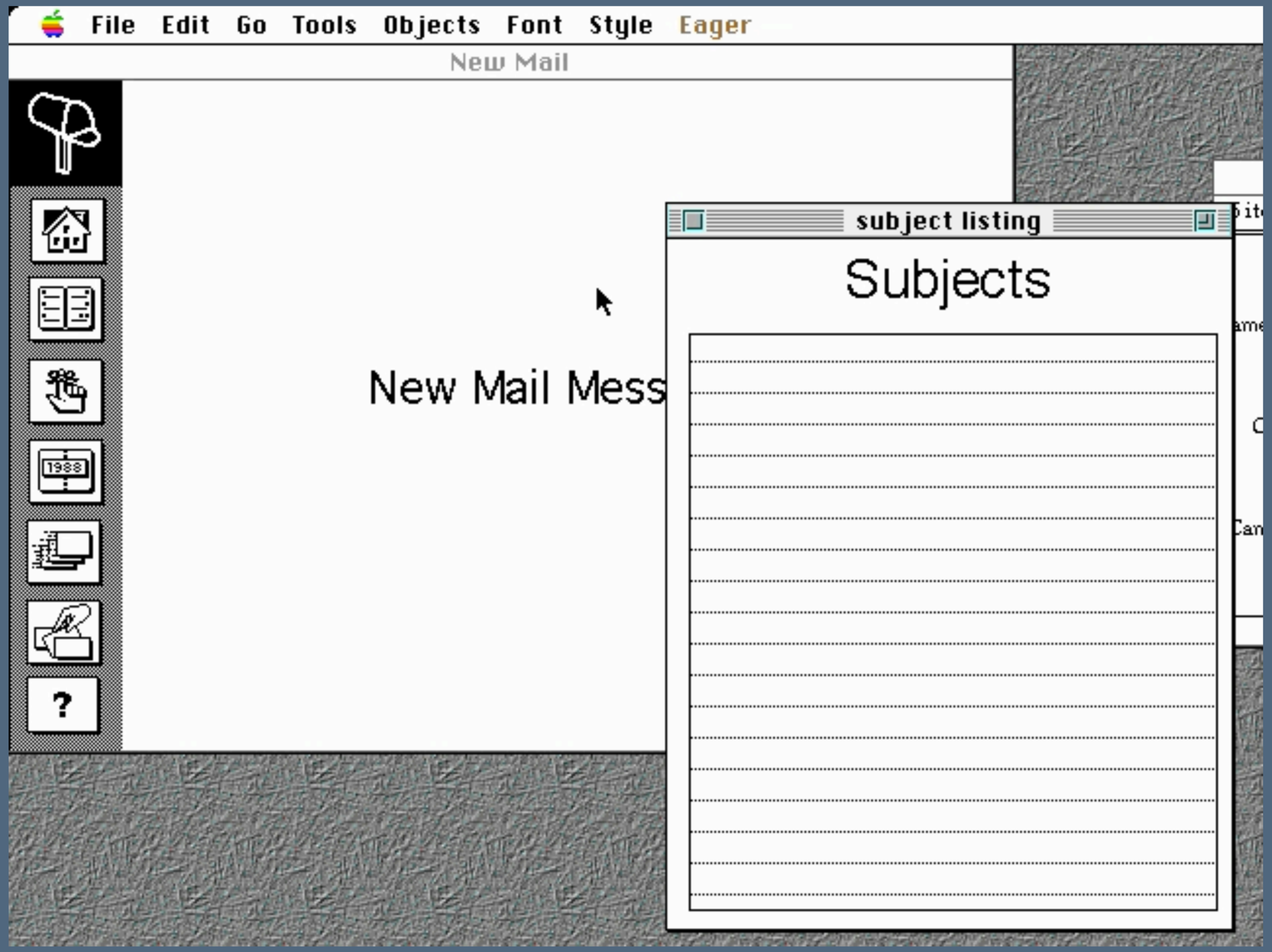
Programming by demonstration (PBD)

Programming by demonstration (PBD): teach a computer a program by doing it yourself while it watches

PBD on the desktop

[Cypher 1991]

Infer a macro by **watching the user's behavior**



Programming by demonstration (PBD)

Programming by demonstration (PBD): teach a computer a program by doing it yourself while it watches

Challenges

- There is an infinite, and hugely branching, space of programs that might be inferred
- Inferred macros can be extremely brittle

Modern PBD: Excel flash fill

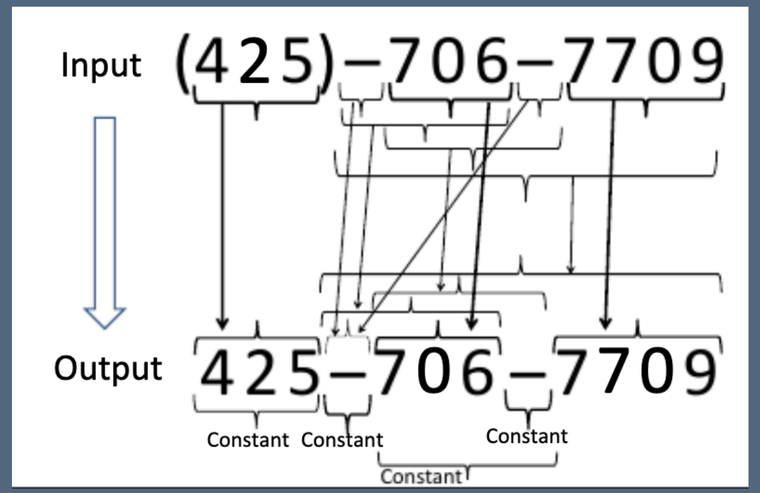
[Gulwani 2011]

Modern PBD: Excel flash fill

[Gulwani 2011]

Develop a domain-specific language of string transformations, and learn from examples how to decompose it into subproblems

Machine learning ranks between all possible valid programs



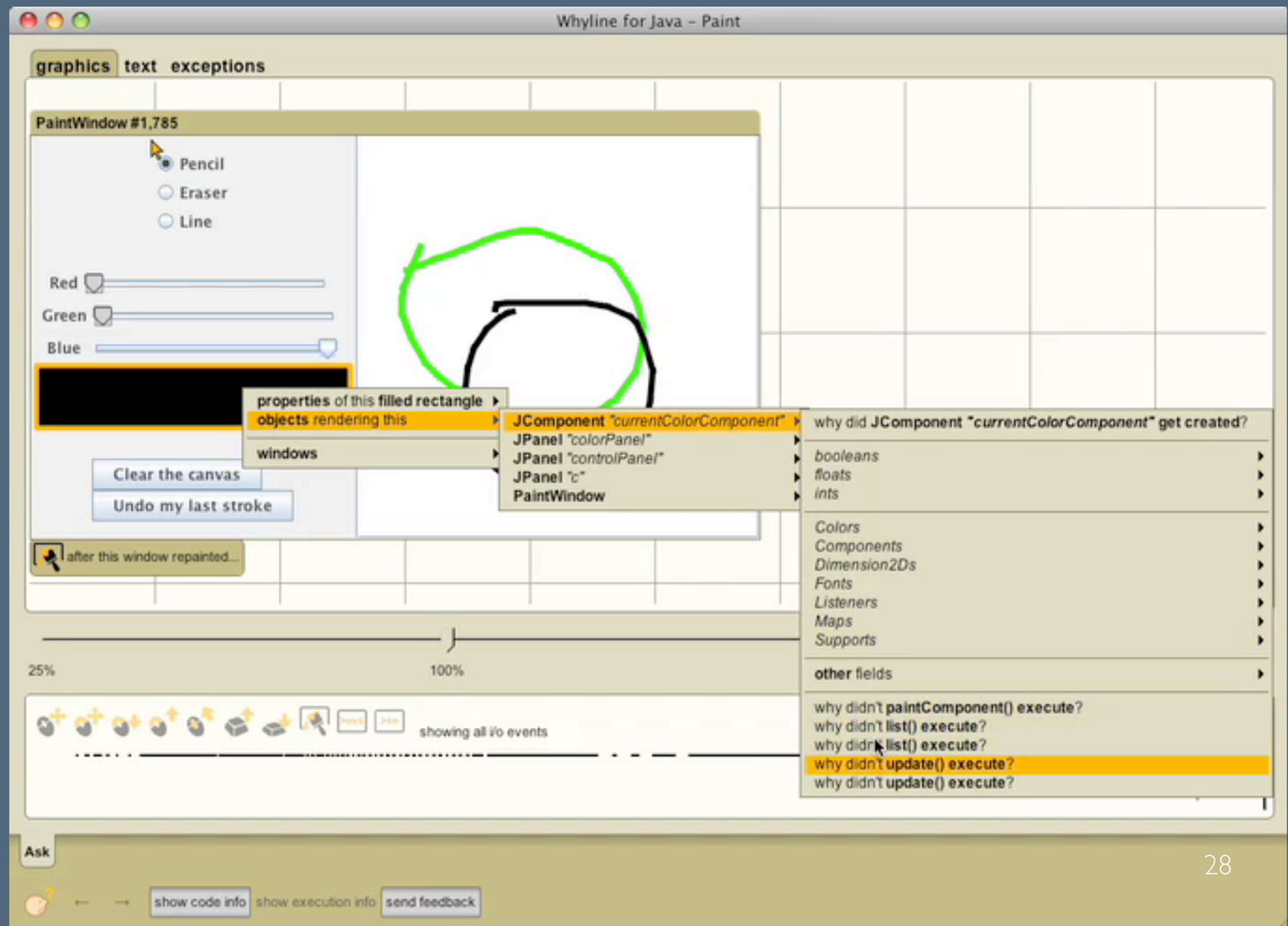
6 S 33rd St, Aston, Delaware, PA, 19014	Gloucester
6 Greenleaf Ave, San Jose, Santa Clara, CA, 95111	Orleans
618 W Yakima Ave, Irving, Dallas, TX, 75062	Livingston
74 S Westgate St, Albany, Albany, NY, 12204	Gloucester
3273 State St, Middlesex, Middlesex, NJ, 8846	Orleans
1 Central Ave, Stevens Point, Portage, WI, 54481	Livingston
86 Nw 66th St #8673, Shawnee, Johnson, KS, 66218	Gloucester

Asking ‘why’ questions of code

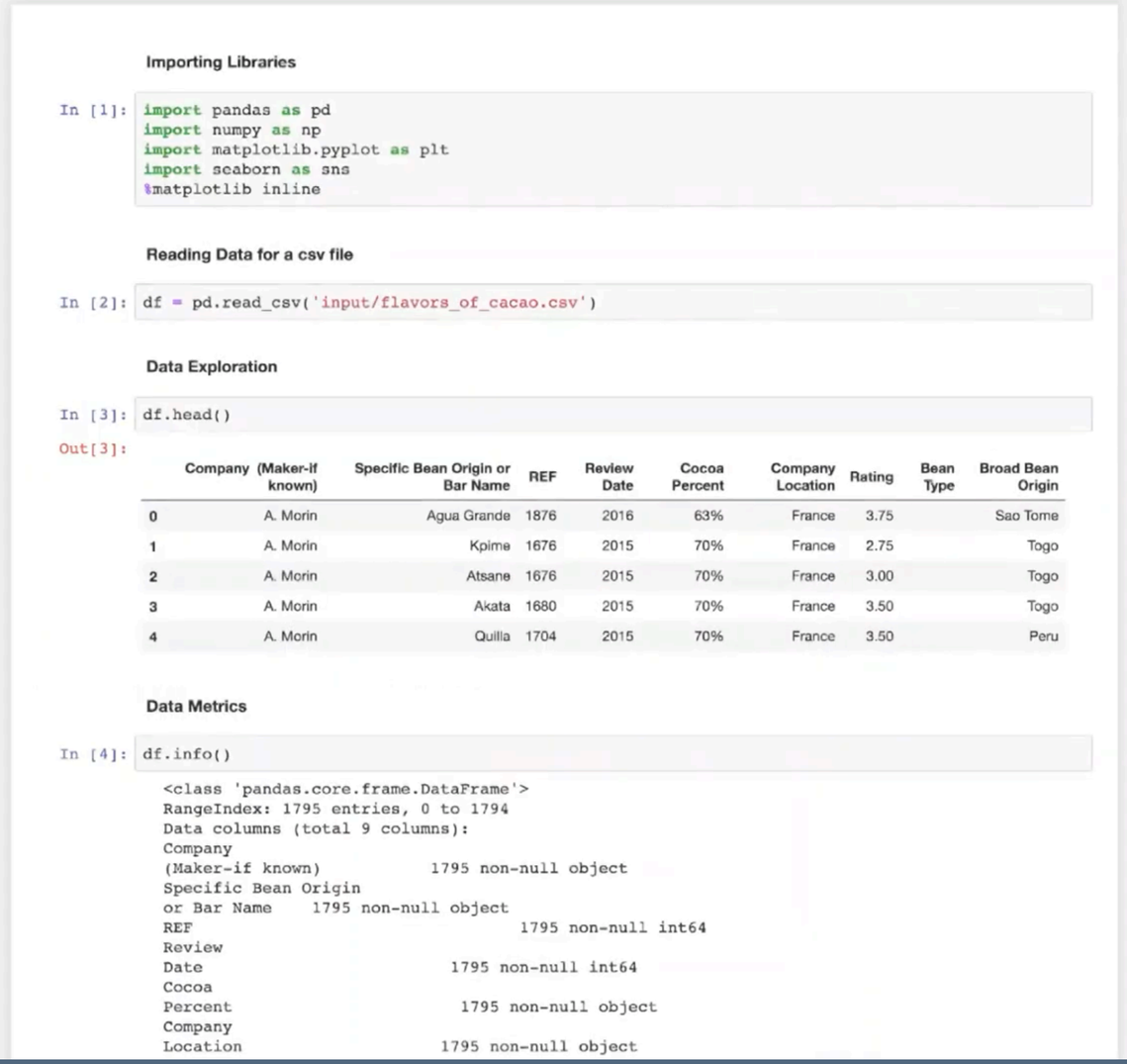
[Ko and Myers CHI ’04, ICSE ’09]

Debugging problems often reduce to “why” questions, but these questions are **challenging to answer** (=high threshold)

Analyze program traces to answer many unanswered “why” and “why not” questions about what just happened



Data science notebooks



Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Reading Data for a csv file

In [2]:
```
df = pd.read_csv('input/flavors_of_cacao.csv')
```

Data Exploration

In [3]:
```
df.head()
```

Out[3]:

	Company (Maker-if known)	Specific Bean Origin or Bar Name	REF	Review Date	Cocoa Percent	Company Location	Rating	Bean Type	Broad Bean Origin
0	A. Morin	Agua Grande	1876	2016	63%	France	3.75		Sao Tome
1	A. Morin	Kpime	1676	2015	70%	France	2.75		Togo
2	A. Morin	Atsane	1676	2015	70%	France	3.00		Togo
3	A. Morin	Akata	1680	2015	70%	France	3.50		Togo
4	A. Morin	Quilla	1704	2015	70%	France	3.50		Peru

Data Metrics

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1795 entries, 0 to 1794
Data columns (total 9 columns):
Company
(Maker-if known)           1795 non-null object
Specific Bean Origin
or Bar Name    1795 non-null object
REF                                     1795 non-null int64
Review
Date                          1795 non-null int64
Cocoa
Percent                        1795 non-null object
Company
Location                      1795 non-null object
```

Automatic cleanup of Jupyter notebooks by tracking provenance across cells [Head et al. 2019]

Raising the ceiling

Goal: increase expressivity—range and (sometimes) complexity of what can be created

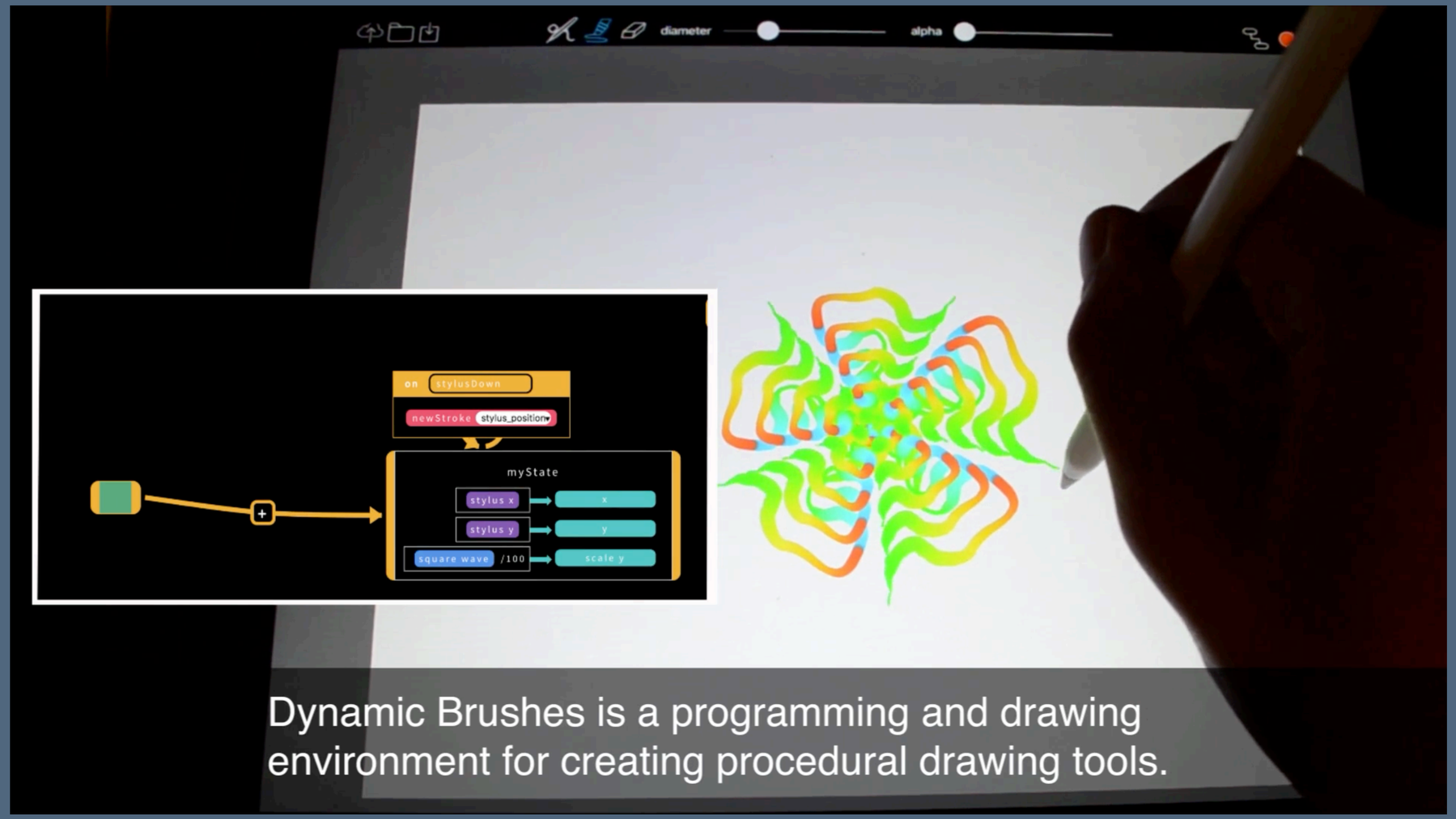
How to increase the ceiling

Identify opportunities for **untapped expressivity** in the current language, and position the software to expose that level of expressivity

This is not about "adding knobs": it's about (metaphorically) providing new paint colors in the palette

Programmable artist brushes

[Jacobs et al. 2018]



Attaching computational functions to brushes enables new forms of artistic expression

Alternative representations for AI under disagreement

[Gordon et al. 2022]

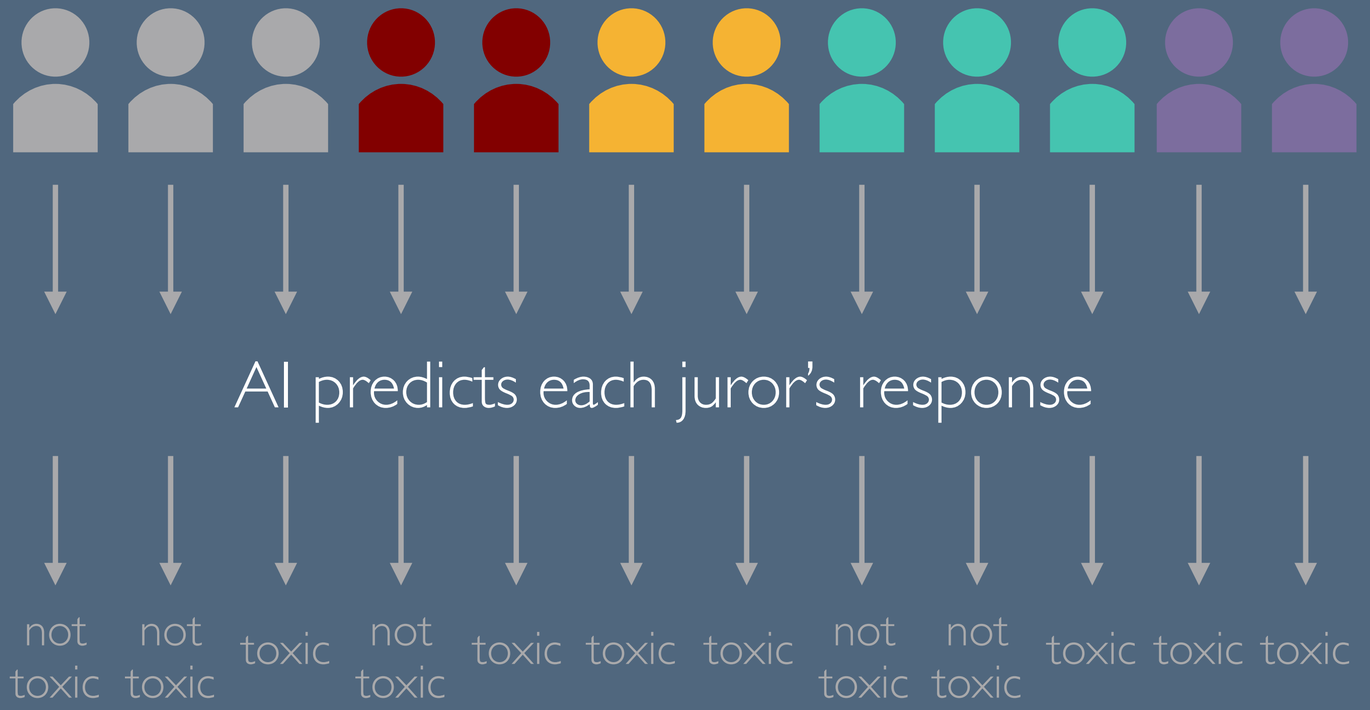
Algorithmic decision as a jury decision. Specify a jury of, say, 12 members, and articulate what proportion of the jury should represent each group and intersectional identity

"For this jury of 50% men and 50% women, which is split evenly between White, Hispanic, AAPI, Black, and Native American jurors, 56% agree the comment is toxic."

Jury learning

"1. People still eat at Pizza Hut? Gross. 2. It is shameful how this country […]"

Input



7 to 5: toxic

Compose a jury by selecting from characteristics in the dataset

Your jury composition Total: 8

A_1 A_2 A_3 A_4 A_5 B_1 B_2 B_3

Juror Selection

+ Add a juror sheet

Juror Sheet A

⊕ Add characteristic

Seats 5

Juror Sheet B

⊕ Add characteristic

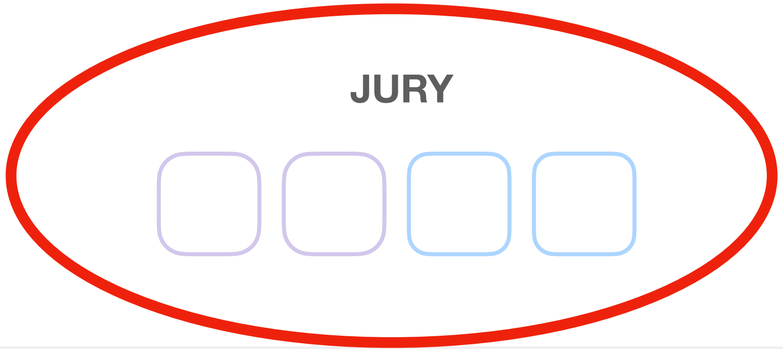
Seats 3

Your input example

Place a comment here that you would like to test

→ View Jury Outcome

Compose a jury by selecting from characteristics in the dataset



Your jury composition Total: 4

A_1 A_2 B_1 B_2

Juror Selection

+ Add a juror sheet

Juror Sheet A

Political affiliation: Liberal

⊕ Add characteristic

Seats: 2

Juror Sheet B

Is Parent ☑

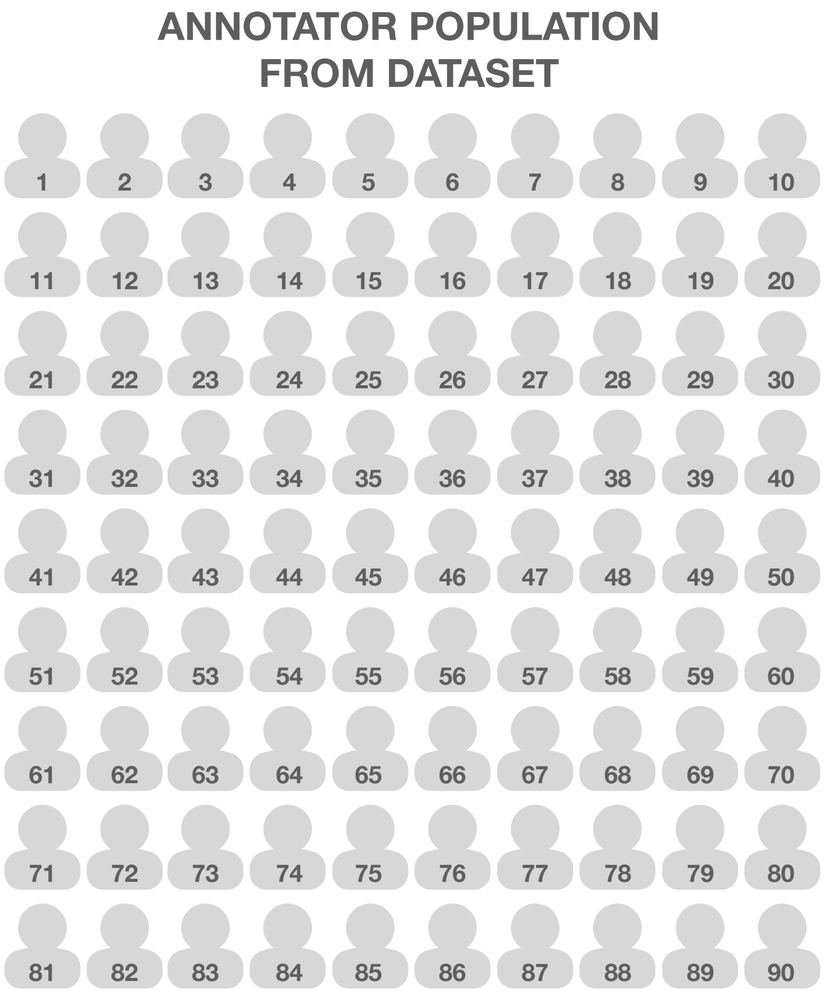
Education: HS Diploma

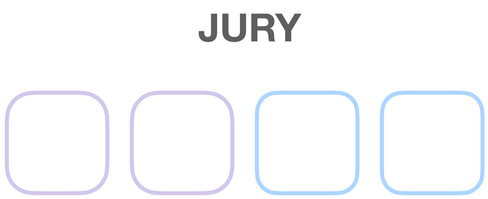
⊕ Add characteristic

Seats: 2

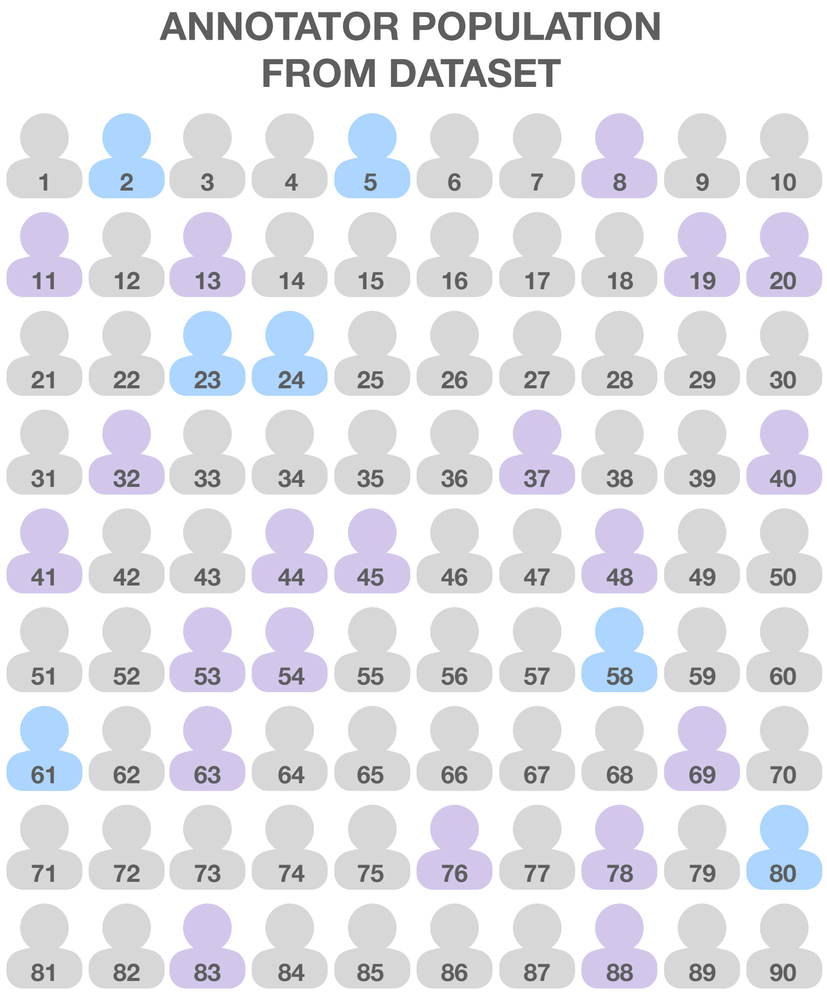
Your input example

System selects matching annotators from the dataset as jurors





System selects matching annotators from the dataset as jurors

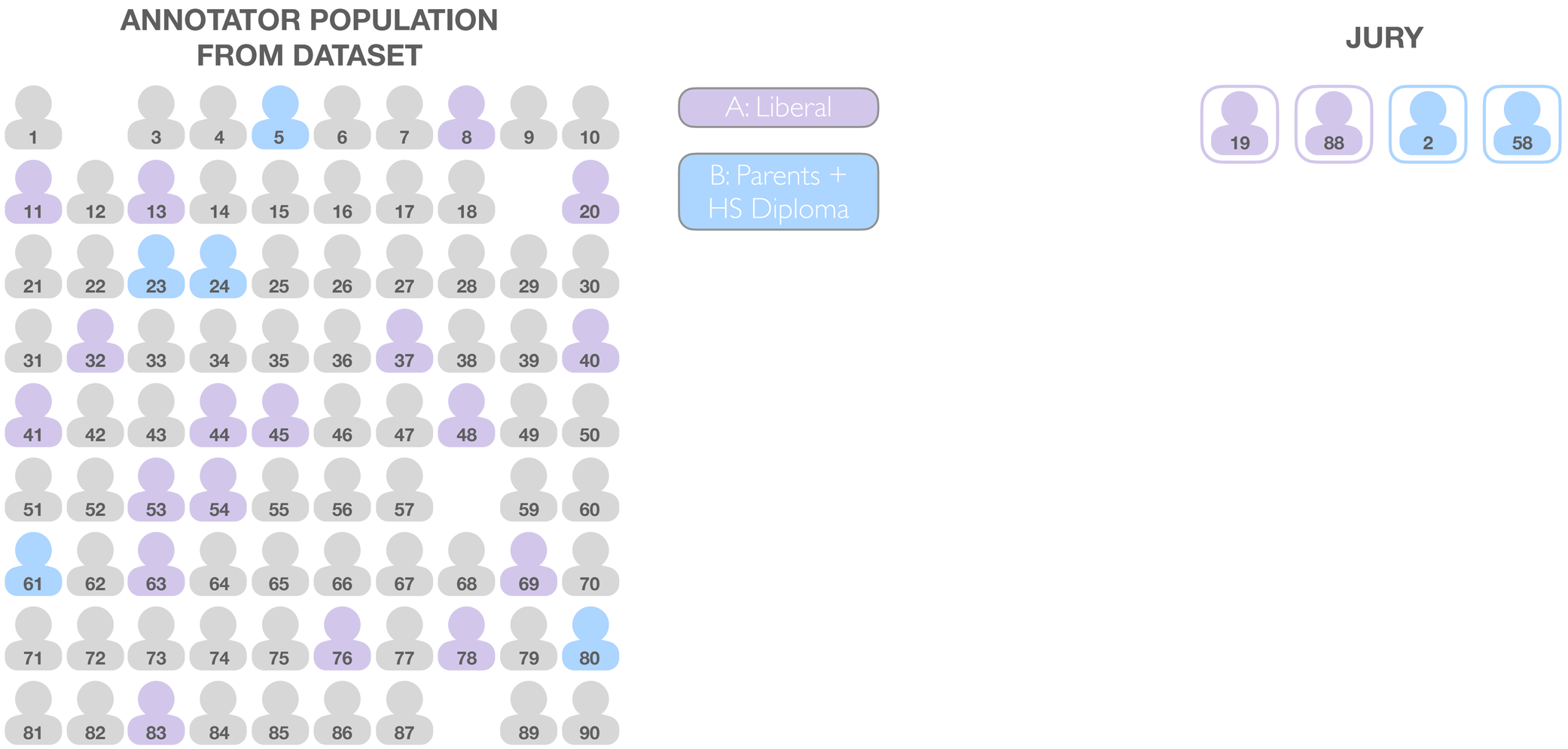


A: Liberal

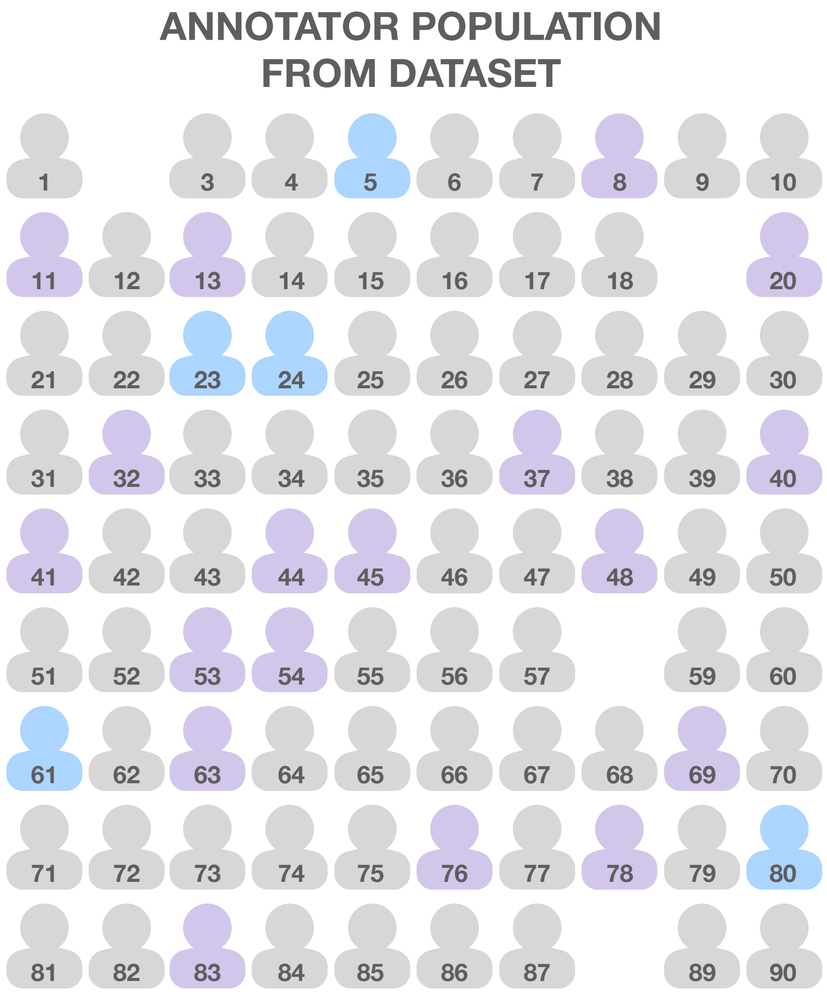
B: Parents + HS Diploma

JURY

System selects matching annotators from the dataset as jurors

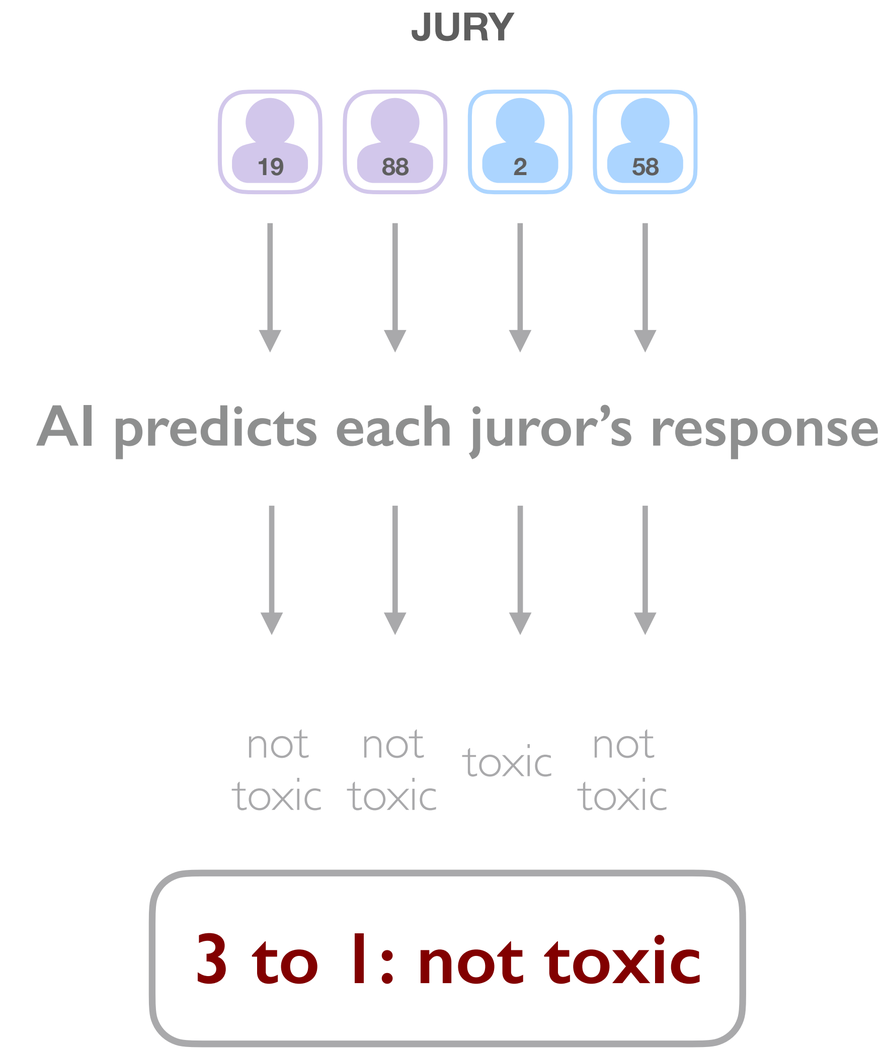


AI predicts how each juror would vote



A: Liberal

B: Parents + HS Diploma



Why does Jury Learning raise the ceiling?

One could argue that jury learning lowers the threshold, since it **refocuses attention from technical desiderata into "who should be on the jury?"**

But, at least as importantly, it **raises the ceiling by increasing the expressivity of the AI models you can construct** for tasks where there is nontrivial disagreement

(This expressivity is why jury learning outperforms SOTA on the task.)

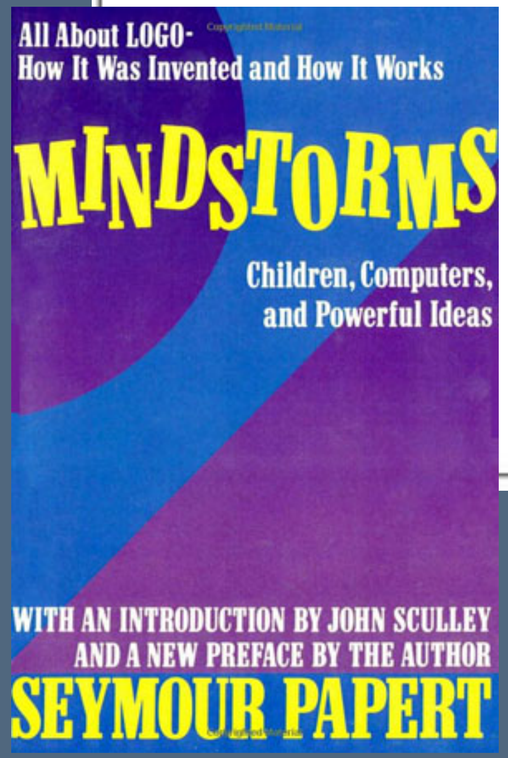
Learning programming

Logo: programming for children

[Papert 1980]

Constructionist learning: a paradigm where learners work on things they care about, guided by their own curiosity, often in social settings.

Lego Mindstorms followed in the mold of Logo and was named after it

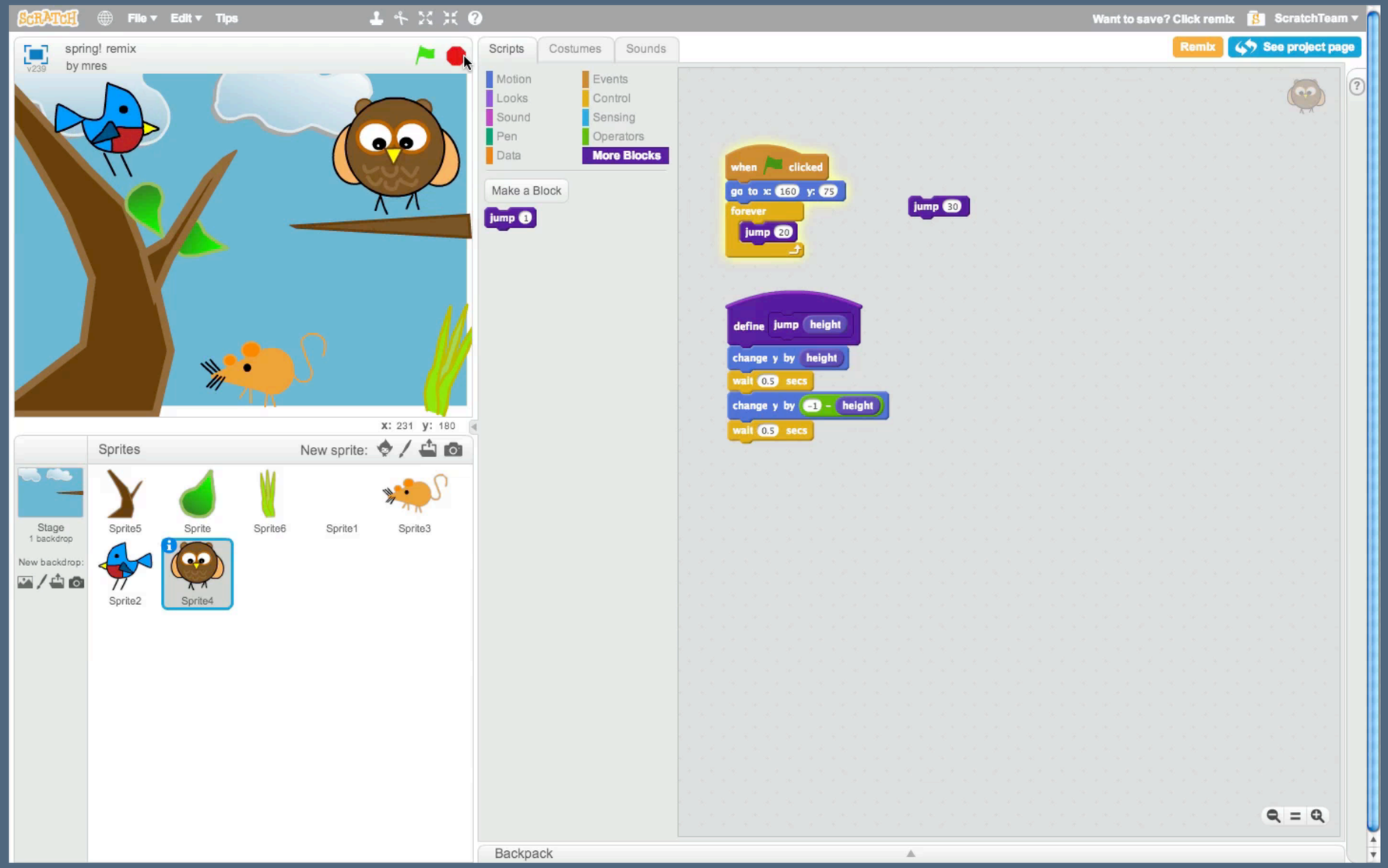


Scratch

[Resnick et al. 2009]

Inherited from Logo:

Block-based programming of simple animations and games as a gateway to programming for children

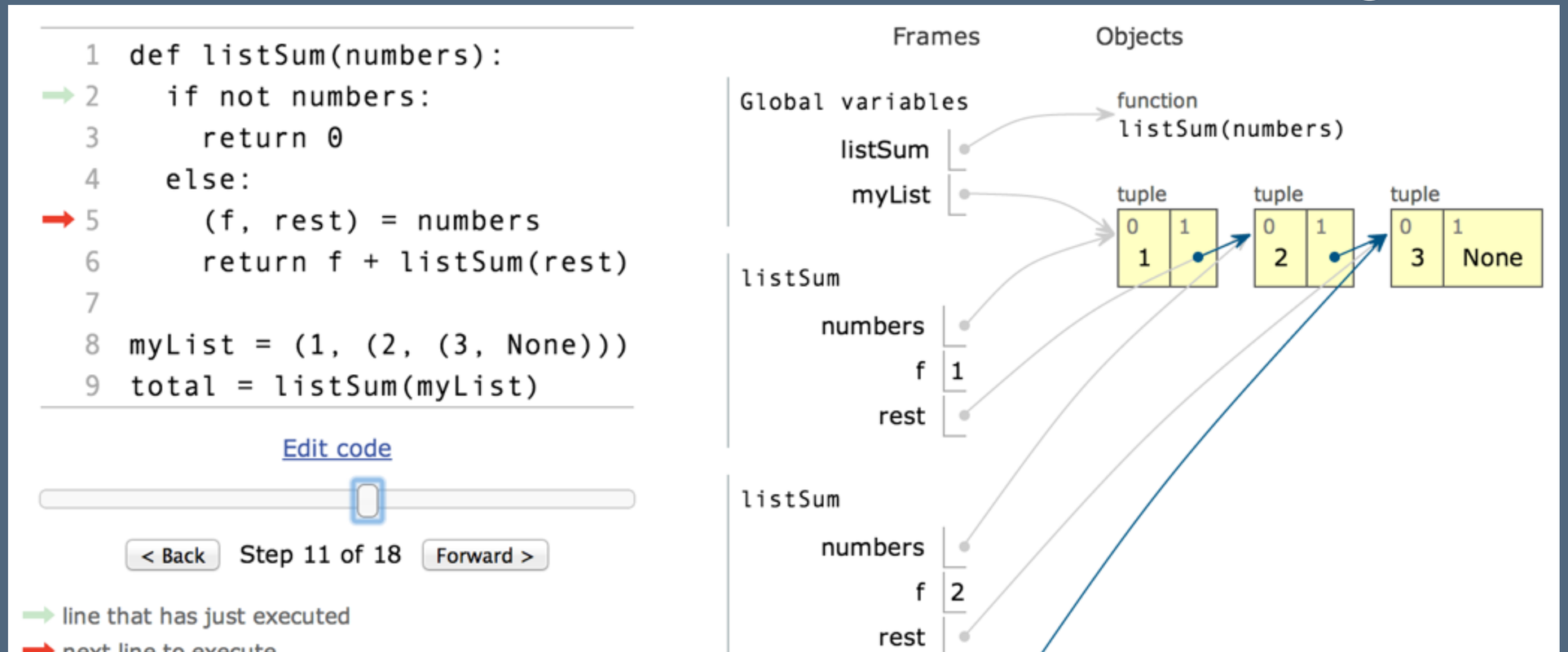


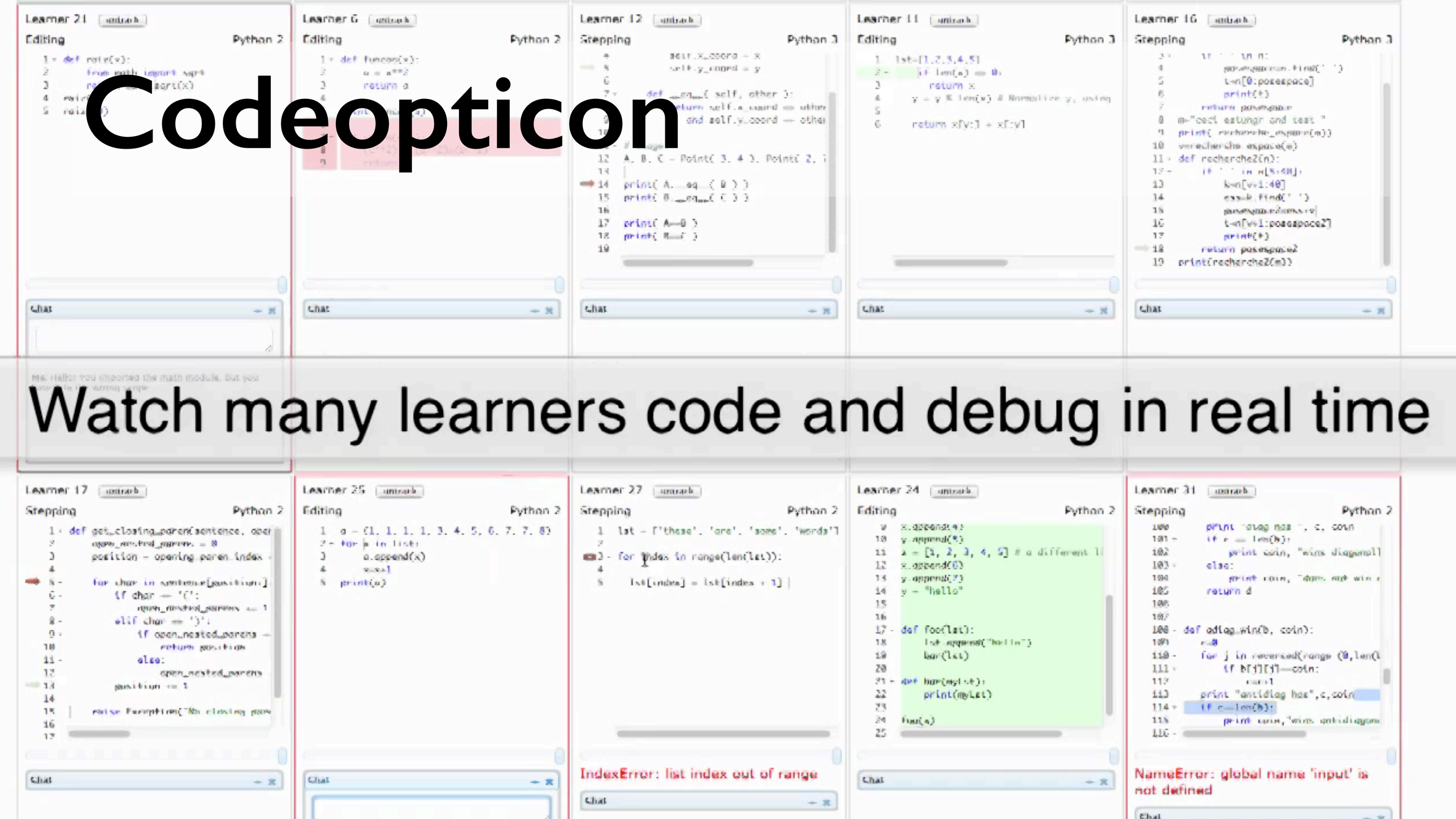
Online python tutor

[Guo 2013]

Embeddable Python data structure visualization

Over 200,000 users and a dozen universities using it




Codeopticon
Watch many learners code and debug in real time
Learner 21
Learner 6
Learner 12
Learner 11
Learner 16
Learner 17
Learner 25
Learner 27
Learner 24
Learner 31
IndexError: list index out of range
NameError: global name 'input' is not defined

Clustering student programs

[Glassman and Miller 2015]

iterPower

done

showing 862 total stacks

that represent 3842 total submissions

filtering by:

nothing yet

Filter | Rewrite | Legend

lines that appear in at least 50 submissions

Largest stack (matching filters)

1534

```
def iterPower(base,exp):
    result=1
    while exp>0:
        result*=base
        exp-=1
    return result
```

Remaining stacks (matching filters)

374

```
def iterPower(base,exp):
    result=1
    while exp>0:
        result=result*base
        exp-=1
    return result
```

153

```
def iterPower(base,exp):
    result=1
    while exp>0:
        result=result*base
        exp=exp-1
    return result
```

77 base=resultB

2592 def iterPower(base,exp):

701 def iterPower(base,expB):

349 def iterPower(base,expC):

51 def iterPower(base,expD):

51 def iterPower(resultB,expC):

55 elif expC==1:

527 else:

2466 exp-=1

279 exp=exp-1

135 exp=expB

366 expC-=1

65 expC=expC-1

63 for i in range(0,expB):

174 for i in range(expB):

52 iC+=1

64 iC=0

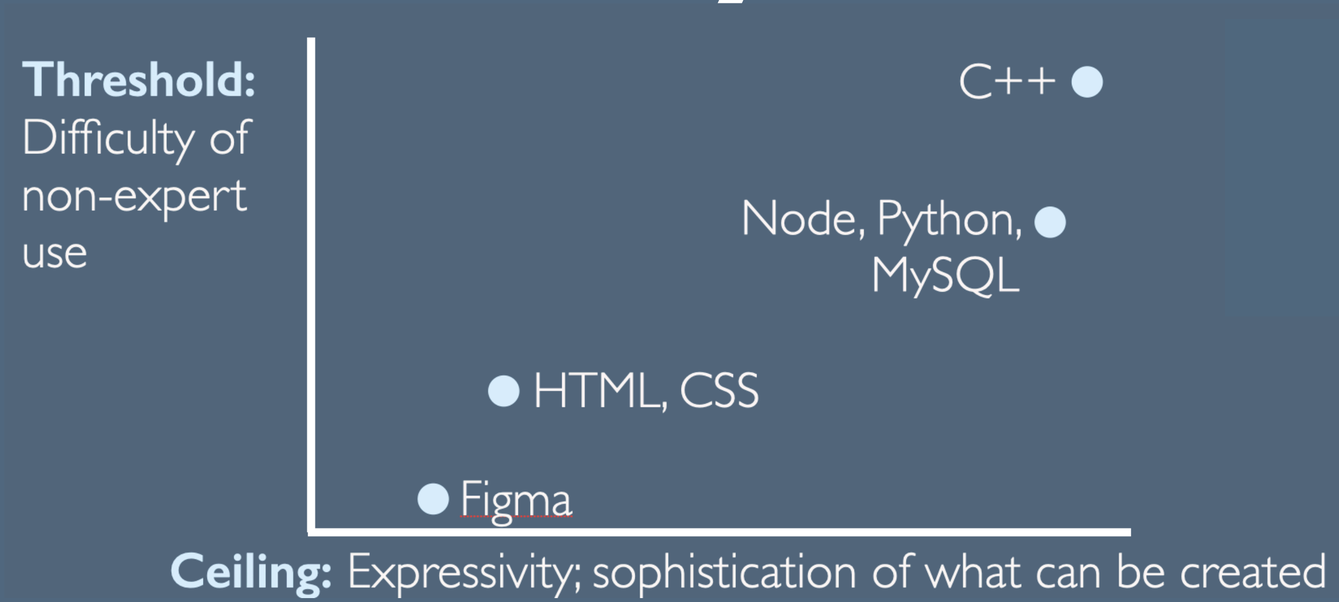
204 if exp==0:

210 if expB==0:

350 if expC==0:

2035 result*=base

Summary



Programming tools often either aim to **reduce the threshold** or **raise the ceiling** — how depends on which one we're pursuing

Successful programming tools **shift our cognitive problem representations** to make the task more readily solvable

Tools for **learning programming** lower thresholds, aim for high ceilings, teach representations, and facilitate instruction.

References

Albaugh, Lea, Scott Hudson, and Lining Yao. "Digital fabrication of soft actuated objects by machine knitting." Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. 2019.

Bertin, Jacques. Semiology of graphics. University of Wisconsin press, 1983.

Brandt, Joel, et al. "Two studies of opportunistic programming: interleaving web foraging, learning, and writing code." Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. 2009.

Cypher, Allen. "Eager: Programming repetitive tasks by example." Proceedings of the SIGCHI conference on Human factors in computing systems. 1991.

Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

Glassman, Elena L., et al. "OverCode: Visualizing variation in student solutions to programming problems at scale." ACM Transactions on Computer-Human Interaction (TOCHI) 22.2 (2015): 1-35.

Gulwani, Sumit. "Automating string processing in spreadsheets using input-output examples." ACM Sigplan Notices 46.1 (2011): 317-330.

Guo, Philip J. "Codeopticon: Real-time, one-to-many human tutoring for computer programming." Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology. 2015.

References

Guo, Philip J. "Online python tutor: embeddable web-based program visualization for cs education." Proceeding of the 44th ACM technical symposium on Computer science education. 2013.

Head, Andrew, et al. "Managing messes in computational notebooks." Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. 2019.

Jacobs, Jennifer, et al. "Extending manual drawing practices with artist-centric programming tools." Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. 2018.

Ko, Amy J., and Brad A. Myers. "Debugging reinvented: asking and answering why and why not questions about program behavior." Proceedings of the 30th international conference on Software engineering. 2008.

Ko, Amy J., Brad A. Myers, and Htet Htet Aung. "Six learning barriers in end-user programming systems." 2004 IEEE Symposium on Visual Languages-Human Centric Computing. IEEE, 2004.

Ko, Amy J., et al. "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks." IEEE Transactions on software engineering 32.12 (2006): 971-987.

Ko, Amy J., Robert DeLine, and Gina Venolia. "Information needs in collocated software development teams." 29th International Conference on Software Engineering (ICSE'07). IEEE, 2007.

References

LaToza, Thomas D., Gina Venolia, and Robert DeLine. "Maintaining mental models: a study of developer work habits." Proceedings of the 28th international conference on Software engineering. 2006.

Mackinlay, Jock. "Automating the design of graphical presentations of relational information." Acm Transactions On Graphics (Tog) 5.2 (1986): 110-141.

Myers, Brad, Scott E. Hudson, and Randy Pausch. "Past, present, and future of user interface software tools." ACM Transactions on Computer-Human Interaction (TOCHI) 7.1 (2000): 3-28.

Norman, Don. Things that make us smart: Defending human attributes in the age of the machine. Basic Books. 1994.

Papert, Seymour. "Mindstorms: children, computers, and powerful ideas." (1980).

Resnick, Mitchel, et al. "Scratch: programming for all." Communications of the ACM 52.11 (2009): 60-67.

Rheingold, Howard. Tools for thought: The history and future of mind-expanding technology. MIT press, 2000.

Satyanarayan, Arvind, et al. "Vega-lite: A grammar of interactive graphics." IEEE transactions on visualization and computer graphics 23.1 (2016): 341-350.

Simon, Herbert A. "The sciences of the artificial." (1969).