CLUSTERING AND VISUALIZING SOLUTION VARIATION IN MASSIVE PROGRAMMING CLASSES

Elena L. Glassman

MIT CSAIL UID GROUP

Advised by Rob Miller

MASSIVE PROGRAMMING CLASSES

- Introductory class sizes
  - ~1500 at UCBerkeley
  - hundreds at MIT
  - Many more complete online programming exercises (edX, Coursera, Khan Academy)

Margaret Hamilton, NASA software developer

Israeli & Palestinian students writing apps and websites and founding businesses together (MIT MEET)

STUDENTS WRITE MULTIPLE SHORT SOLUTIONS PER WEEK

Problem

1. exponentiate a number

2. compute a deriviative

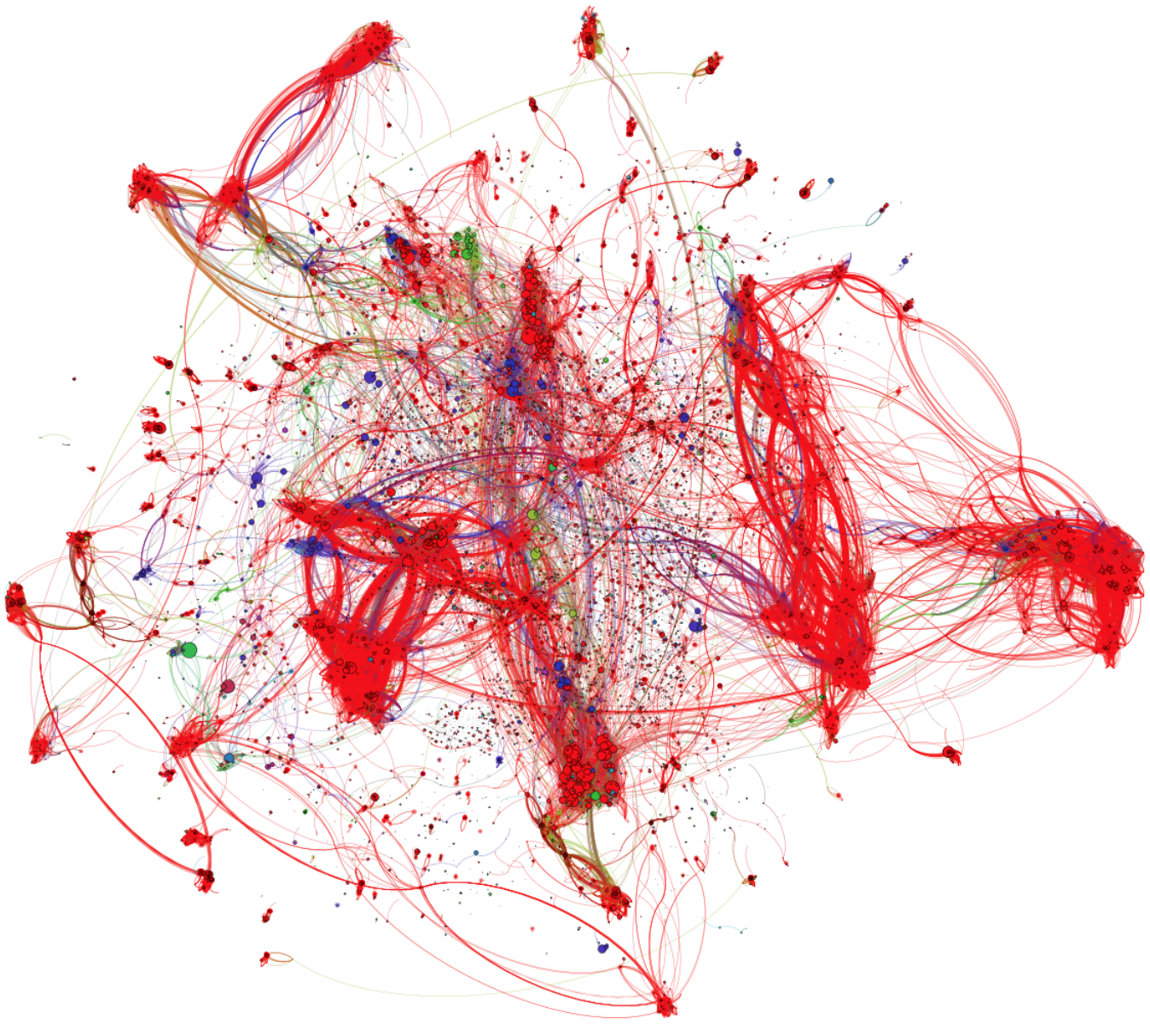
3. compute a logarithm

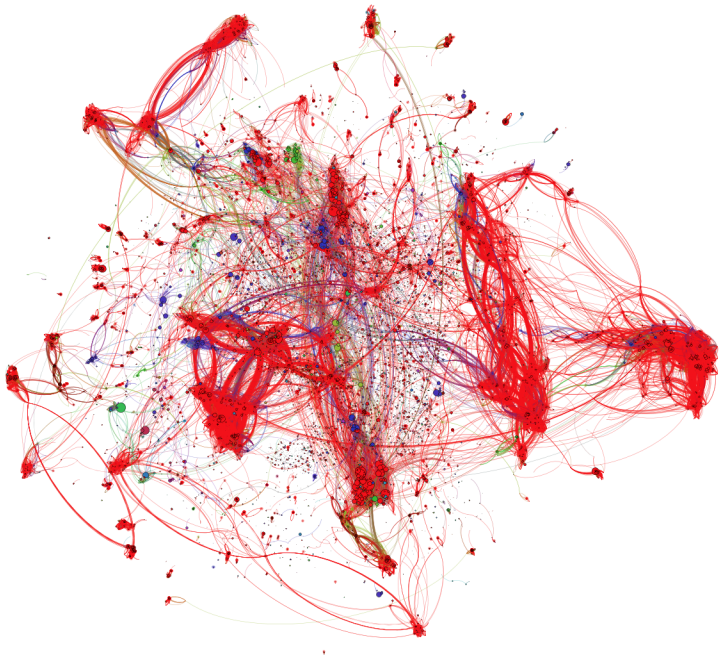
One student solution

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

```
def computeDeriv(poly):
    result=[]
    for i in range(1,len(poly)):
        result.append(float(poly[i]*i))
    return result
```

```
def myLog(x,b):
    power=0
    while b**power<=x:
        power+=1
    return power-1
```





- "Codewebs" by Huang et. al. (WWW '14)
  - ~40,000 student submissions to the same problem
  - Source: Coursera's Machine Learning course
  - Ran on a computing cluster

THESIS CONTRIBUTION 1:

CLUSTER AND VISUALIZE IN A HUMAN-READABLE WAY

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

This synthesized solution represents 1538 solutions

OUTLINE

- Solution Variation (**SV**)
- Research Questions
  - (R1) understand **SV**
  - (R2) feedback on **SV** at scale
  - (R3) personalized feedback on **SV** at scale
  - (R4) collect and distribute student advice, given **SV**
- Discussion

SOLUTION VARIATION

- Correctness
- Approach
- Readability

CORRECTNESS WITH RESPECT TO TEST CASES

1. `power(5,3)` should return `125`
2. `power(3,4)` should return `81`
3. `power(2,5)` should return `32`
4. `...`

(automated with an "autograder")

AUTOGRADERS CAN ONLY DO SO MUCH

These are all "correct":

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

def power(base,exp):
    if exp == 0:
        return 1
    else:
        return base * power(base,
 exp-1)
```

```
def power(base, exp):
    tempBase=base
    result = base

    if type(base)==int:
        while exp==0:
            result = 1
            print(result)
            break
        exp=exp-1
        while exp >0:
            tempCal=abs(tempBase)
            exp=exp-1
            while exp<0:
                break
            for i in range (1,tem
```

APPROACH

- Disregards teacher's request to write it themselves

```
def power(base, exp):
  return base**exp
```

- Reveals possible misconceptions, e.g., unnecessary statements

```
def power(base,exp):
 result=1
 while exp>0:
     result=result*base
     exp-=1
     continue #keyword here does not change execution
 return result
```

APPROACH

- Common

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

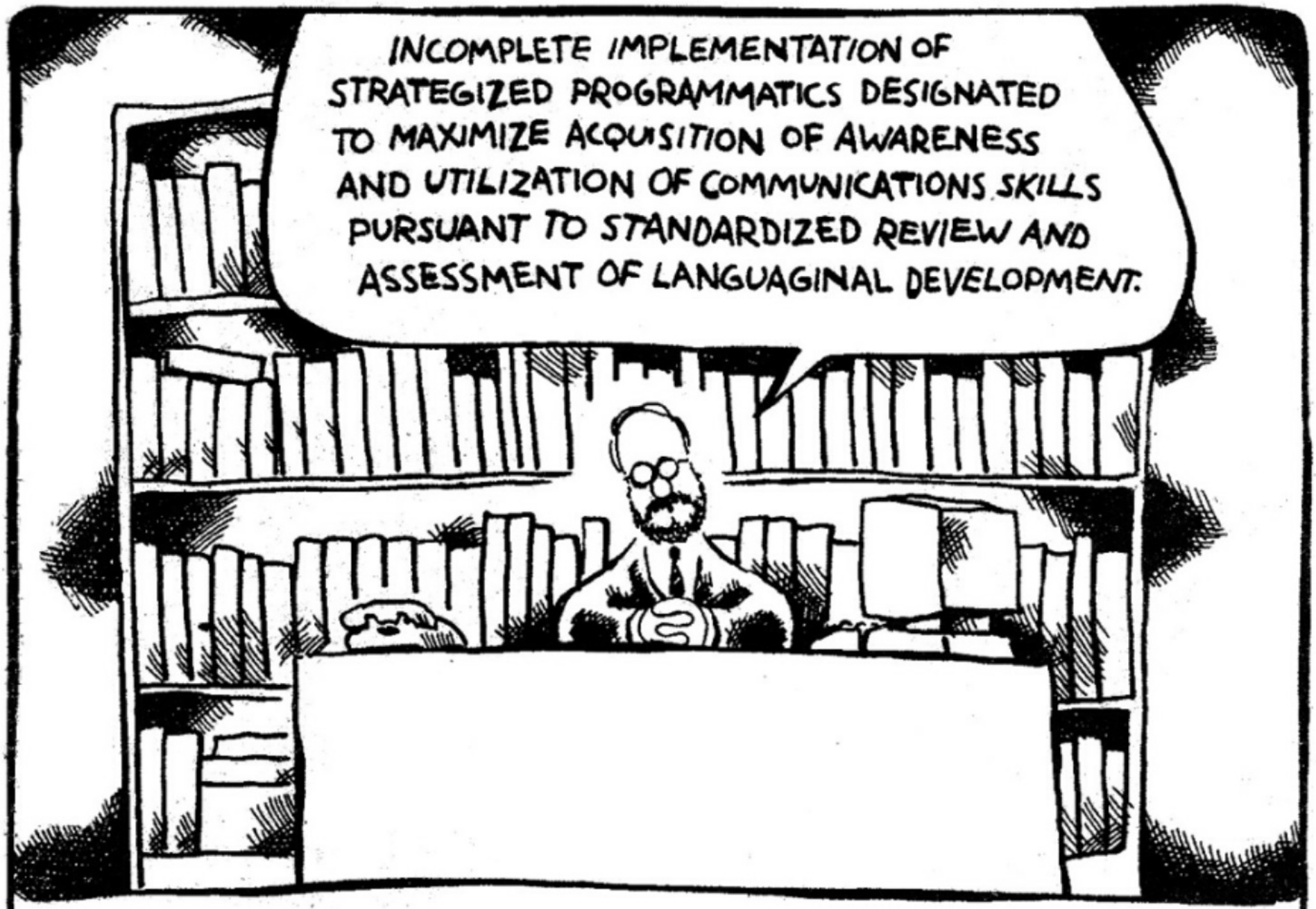
- Unusual, possibly innovative

```
def power(base,exp):
  if exp == 0:
      return 1
  else:
      return base * power(base, exp-1)
```

READABILITY

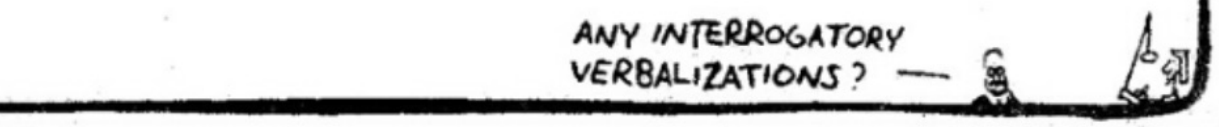
- Correct w.r.t. test cases
- Difficult to read

```
def power(base, exp):
  tempBase=base
  result = base

  if type(base)==int:
      while exp==0:
          result = 1
          print(result)
          break
      exp=exp-1
      while exp >0:
          tempCal=abs(tempBase)
          exp=exp-1
          while exp<0:
              break
          for i in range (1,tempCal):
```



The reason Verbal SAT scores are at an all-time low

The reason verbal SAT scores are at an all time low.



READABILITY MATTERS

- **Student design choices** affect correctness, approach and readability
- Examples:
  - `for` vs. `while`
  - `a *= b` vs. `a = a*b`
  - recursion vs. iteration

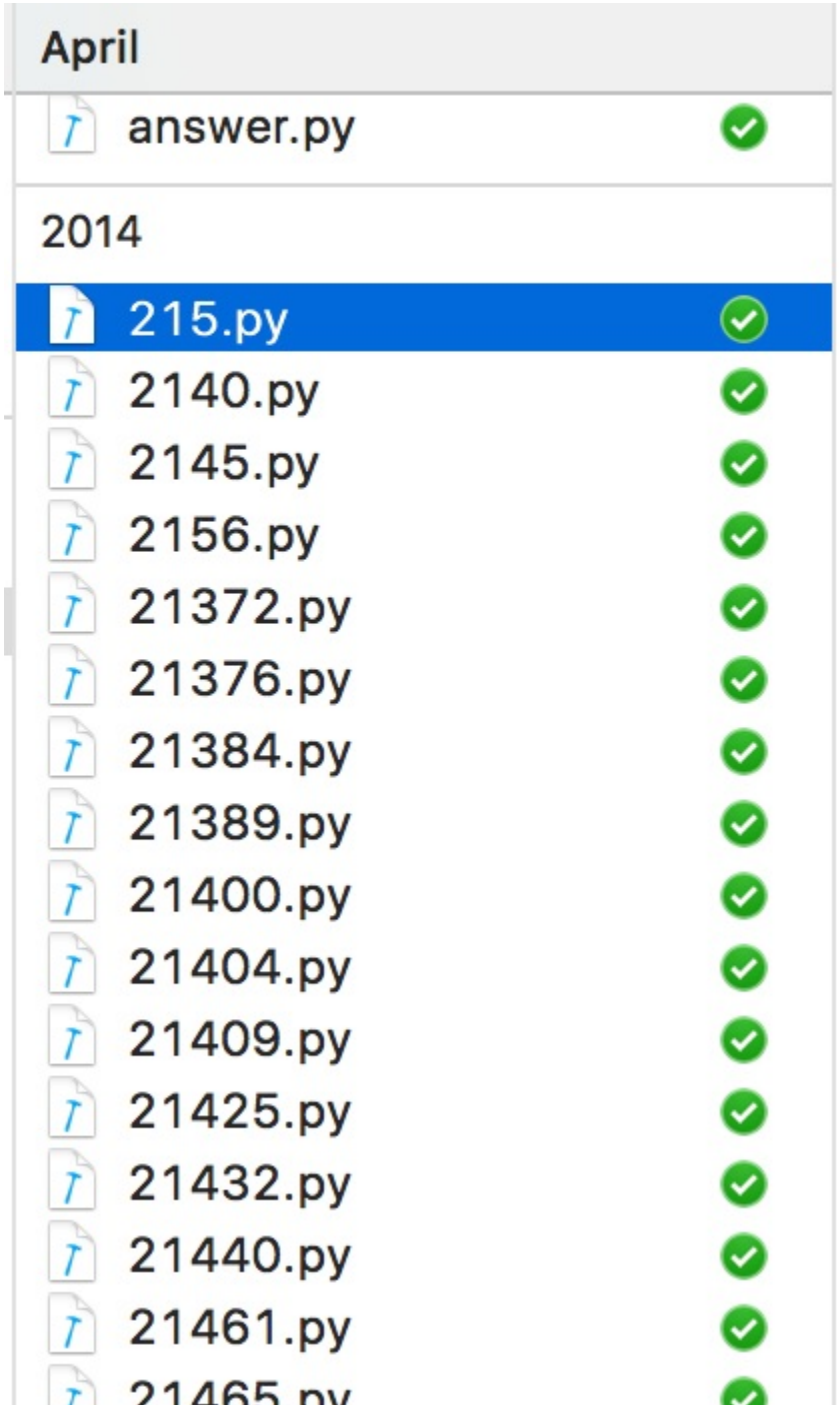
Solution variation is a result of these choices

SOLUTION VARIATION

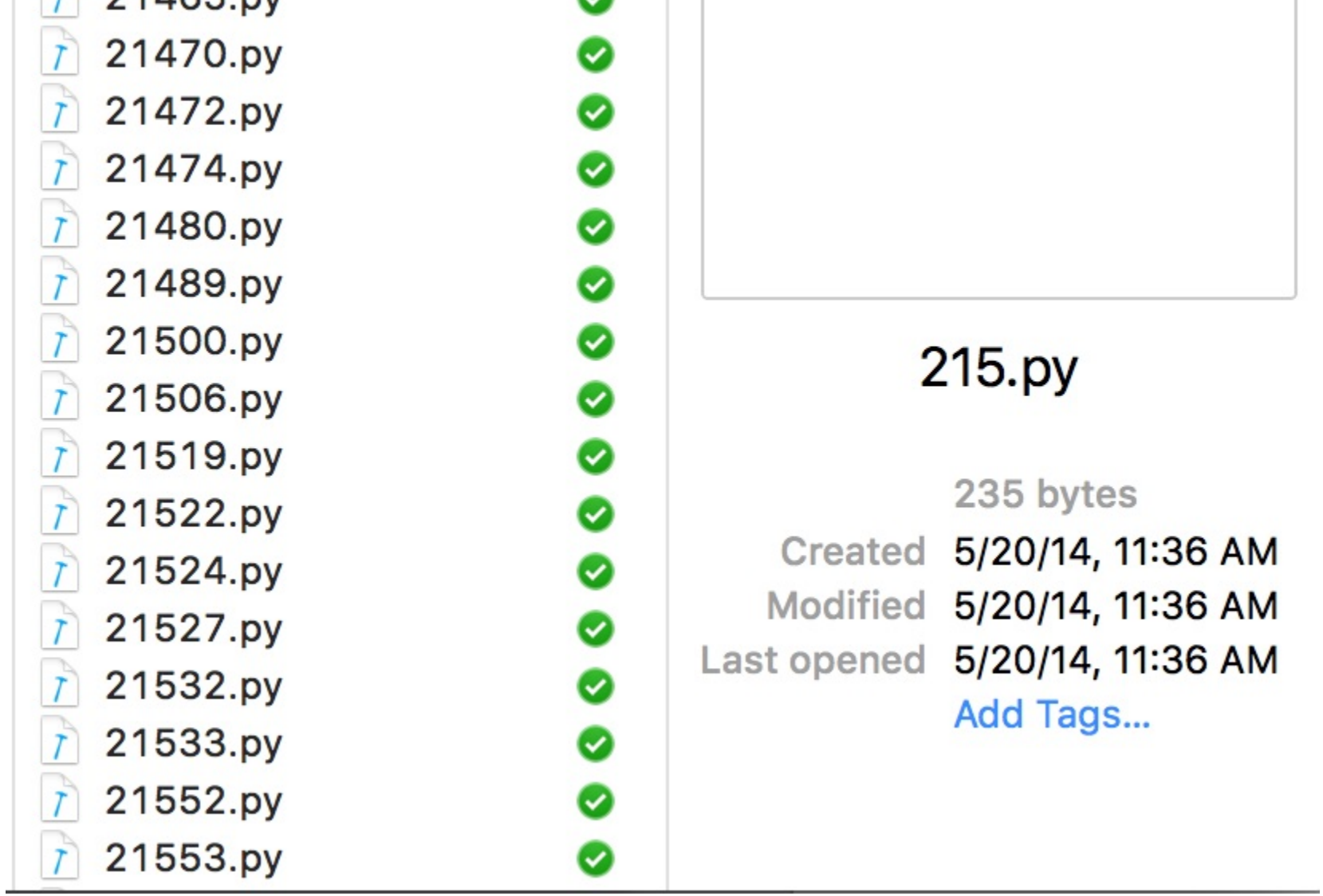
- Different comments, statement order, variable names

```
def iterPower(base, exp):
  '''
  base: int or float.
  exp: int >= 0

  returns: int or float, base^exp
  '''
  result = 1
  while exp > 0:
      result *= base
      exp -= 1
  return result
```

```
def iterPower(base, exp):
  wynik = 1
  while exp > 0:
      exp -= 1  #using exp argument as counter
      wynik *= base
  return wynik
```




```
def iterPower(base, exp):
    '''
    base: int or float.
    exp: int >= 0

    returns: int or
float, base^exp
    '''
    # Your code here
    res=1.
    while exp>0:
        exp-=1
        res=res*base
    return res
```


21470.py
21472.py
21474.py
21480.py
21489.py
21500.py
21506.py
21519.py
21522.py
21524.py
21527.py
21532.py
21533.py
21552.py
21553.py
215.py
235 bytes
Created 5/20/14, 11:36 AM
Modified 5/20/14, 11:36 AM
Last opened 5/20/14, 11:36 AM
Add Tags...

CHALLENGE

How do you understand
(1) what students wrote
(2) give feedback at scale?

OPPORTUNITY

What value *only* exists in a massive programming class?

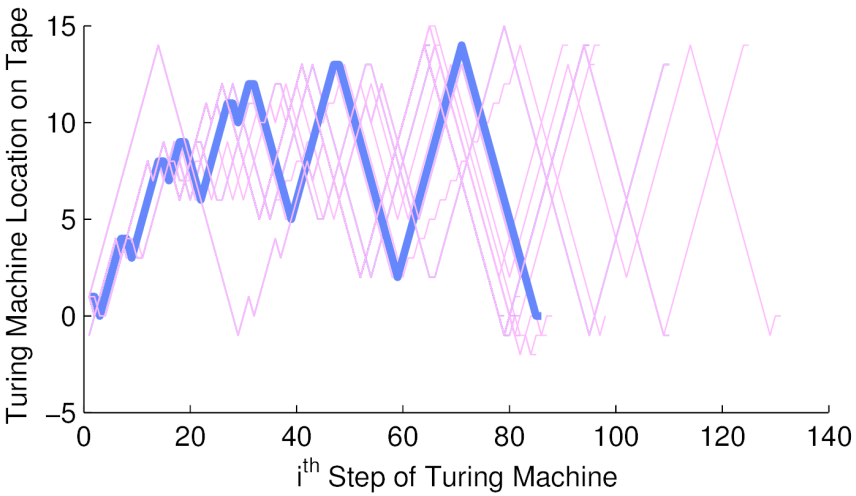
OPPORTUNITY:

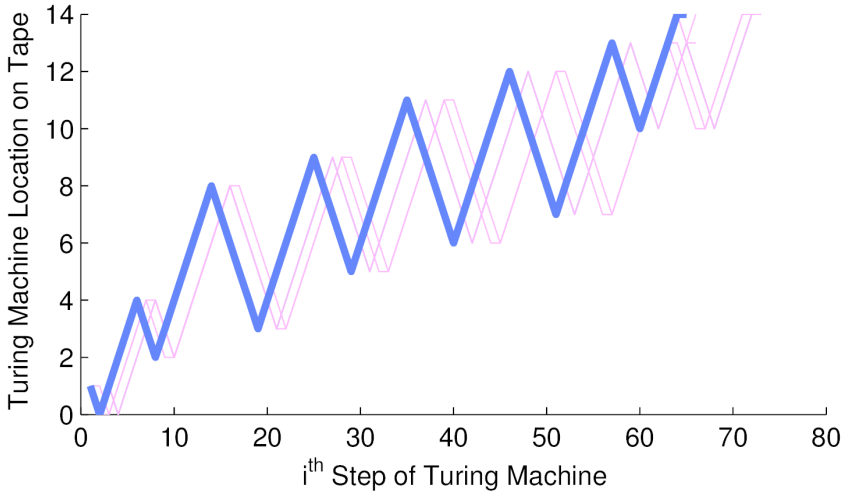
EXPLOITING SOLUTION VARIATION WITHIN LARGE DATASETS

- Teachers could
  - learn about new solutions from students
  - find better examples for discussion
- Teachers could write better feedback, test cases, evaluation rubrics
- Students could be tapped as experts on their solutions (and bugs they fix)

A Turing Machine - Overview

TWO COMMON STRATEGIES





Strategy A
(~50% of correct solutions)

Strategy B
(~40% of correct solutions)

2 solutions passed all test cases
but subverted teacher instructions

- Fellow teachers were not aware
  - of multiple solutions
  - that the test suite was insufficient

OUTLINE

- Solution Variation (SV)
- Research Questions
  - (R1) understand **SV**
  - (R2) feedback on **SV** at scale
  - (R3) personalized feedback on **SV** at scale
  - (R4) collect and distribute student advice, given **SV**
- Discussion

RESEARCH QUESTIONS

When the teacher cannot read all the solutions in a collection,

- (R1) ... how can the teacher understand the common and uncommon variation in their students' solutions?
- (R2) ... how can the teacher give feedback on the approach & readability at scale?
- (R3) ... in a personalized way?
- (R4) How can students do the same?

OUTLINE

- Solution Variation (SV)
- Research Questions
  - ⇒ (R1) understand **SV**
  - (R2) feedback on **SV** at scale
  - (R3) personalized feedback on **SV** at scale
  - (R4) collect and distribute student advice, given **SV**
- Discussion

def power(base, exp):

```
'''iterative so
l'''
result=1 #initia
lize
while exp>0:
  exp-=1 #input
 as counter
  result=result\
         *base
return result
```

```
result=1
while exp>0:
  exp-=1
  result=result\
         *base
return result
```

```
result = 1

while(exp>0):
  exp-=1
  result=result\
         *base

return result
```

```
myans=1
while exp>0:
  myans=myans\
         *base
  exp-=1
return myans
```

```
res=1
while exp>0:
  res=res*base
  exp-=1
return res
```

```
ans = base
if (exp == 0):
  return round(base/base, 4)
else:
  for n in range(exp-1):
    ans *= base
  return round(ans, 4)
```

```
result = base
if exp==0:
  return 1
if exp==1:
  return result
while exp > 1:
  result *= base
  exp -= 1
return result
```

HOW DOES A TEACHER READ THOUSANDS OF SOLUTIONS?

(WITHOUT ACTUALLY READING ALL OF THEM)

SYNTHESIZE SOLUTIONS THAT REPRESENT MANY

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

represents 1538/3852 solutions

PIPELINE

1. Remove comments
2. Standardize formatting
3. Normalize variable names *using student-chosen names across all solutions*

The set of unique solutions (ignoring statement order) at the end of this process are our representatives.

def power(base, exp):

```
'''iterative so
l'''
result=1 #initia
lize
while exp>0:
  exp-=1 #input
 as counter
  result=result\
         *base
return result
```

```
result=1
while exp>0:
  exp-=1
  result=result\
         *base
return result
```

```
result = 1

while(exp>0):
  exp-=1
  result=result\
         *base

return result
```

def power(base, exp):

```
result=1
while exp>0:
  exp-=1
  result=result\
         *base
return result
```

def power(base, exp):

3

```
result=1
while exp>0:
  exp-=1
  result=result\
         *base
return result
```

def power(base, exp):

3

```
result=1
while exp>0:
  exp-=1
  result=result\
         *base
return result
```

```
res=1
while exp>0:
  res=res*base
  exp-=1
return res
```

NORMALIZE VARIABLE NAMES WITH PASSIVE CROWDSOURCING

- Give variables their most popular name across all solutions

PROBLEM: IDENTIFYING VARIABLES ACROSS SOLUTIONS

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

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

1. How do we tell that the variable 'res' (left) is the “same” as the variable 'result' (right)?
2. How do we tell if two students voted for the same variable name?
3. What does the “same variable in two different solutions” even mean?

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

base: 5
exp: 3, 2, 1, 0
result: 1, 5, 25, 125
```

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

base: 5
exp: 3, 2, 1, 0
res: 1, 5, 25, 125
```

Run all on test case(s):

`iterPower(5,3)`

VARIABLES ACROSS ALL SOLUTIONS

- Variable that takes on sequence

```
1, 5, 25, 125
```

  - Occurs in 3081/3842 solutions
  - Most common name

```
result
```

  - Other names

```
wynik, out, total, ans, acum, num, mult
```

- Refer to it as the 'result **common variable**'

VARIABLES ACROSS ALL SOLUTIONS

- Variable that takes on sequence

```
3, 2, 1, 0
```

  - Occurs in 2744/3842 solutions
  - Most common name

```
exp
```

  - Other names

```
count, temp, exp3, exp2, exp1, inexp, old
_exp
```

- Refer to it as the 'exp **common variable**'

PRESERVE 1-TO-1 MAPPING BETWEEN NAME AND BEHAVIOR

Common Variable	Found in	Most common name	Renamed
1, 5, 25, 125	3081 solutions	result	result
5, 25, 125	< 3081 solutions	result	$result_2$

def power(base, exp):

3

```
result=1
while exp>0:
  exp-=1
  result=result\
         *base
return result
```

```
result=1
while exp>0:
  result=result\
         *base
  exp-=1
return result
```

```
result₂ = base
if (exp₂ == 0):
  return round(base/base, 4)
else:
  for n in range(exp₂-1):
    result₂ *= base
  return round(result₂, 4)
```

```
result=1
while exp>0:
  result=result\
         *base
  exp-=1
return result
```

```
result₂ = base
if exp==0:
  return 1
if exp==1:
  return result
while exp > 1:
  result₂ *= bas
e
  exp -= 1
return result₂
```

def power(base, exp):

3

```
result=1
while exp>0:
  exp-=1
  result=result\
         *base
return result
```

```
result=1
while exp>0:
  result=result\
         *base
  exp-=1
return result
```

```
result=1
while exp>0:
  result=result\
         *base
  exp-=1
return result
```

def power(base, exp):

5

```
result=1
while exp>0:
  exp-=1
  result=result\
         *base
return result
```

```
result2 = base
if (exp2 == 0):
  return round(base/base, 4)
else:
  for n in range(exp2-1):
    result2 *= base
  return round(result2, 4)
```

```
result2 = base
if exp==0:
  return 1
if exp==1:
  return result
while exp > 1:
  result2 *= bas
e
  exp -= 1
return result2
```

SYNTHESIZED REPRESENTATIVE SOLUTIONS

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

represents 1538 solutions

SYNTHESIZED REPRESENTATIVE SOLUTIONS

1538

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

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
```

88

```
def iterPower(base,exp2):
```

```
result=1
for i in range(exp2):
    result*=base
```

SYNTHESIZED REPRESENTATIVE SOLUTIONS

97

```
def getGuessedWord(secretWord,lettersGuessed):
    result=''
    for letter in secretWord:
        if letter in lettersGuessed:
            result+=letter
        else:
            result+='_ '
    return result
```

36

```
def getGuessedWord(secretWord,lettersGuessed):
    result=''
    for letter in secretWord:
        if letter in lettersGuessed:
            result=result+letter
        else:
            result=result+'_ '
    return result
```

15

```
def getGuessedWord(secretWord,lettersGuessed):
    result=""
    for letter in secretWord:
        if letter in lettersGuessed:
            result+=letter
        else:
            result+="_ "
    return result
```

14

```
def getGuessedWord(secretWord,lettersGuessed):
    result=''
    for letter in secretWord:
        if letter not in lettersGuessed:
            result+='_ '
        else:
```

SYNTHESIZED REPRESENTATIVE SOLUTIONS

22

```
def computeDeriv(poly):
    if len(poly)==1:
        return[0.0]
    result=[]
    for i in range(1,len(poly)):
        result.append(float(poly[i]*i))
    return result
```

21

```
def computeDeriv(poly):
    result=[]
    evaluatedTerm=0
    i2=0
    for i2 in range(len(poly)):
        evaluatedTerm=round(poly[i2]*i2,2)
        if i2>=1:
            result.append(evaluatedTerm)
    return result
```

14

```
def computeDeriv(poly):
    result=[]
    for i in range(1,len(poly)):
        result.append(float(poly[i]*i))
    return result
```

13

```
def computeDeriv(poly):
    result=[]
    i4=0
    for i3 in poly:
        if i4>0:
            result=result+[float(i3*i4)]
```

SYNTHESIZED REPRESENTATIVE SOLUTIONS

37

```
def myLog(x,b):
    power=0
    while b**power<=x:
        power+=1
    return power-1
```

6

```
def myLog(x,b):
    power2=0
    while b**(power2+1)<=x:
        power2+=1
    return power2
```

5

```
def myLog(x,b):
    i3=1
    while b**i3<=x:
        i3+=1
    return i3-1
```

3

```
def myLog(x,b):
    power3=0
    while b**power3<=x:
        power3+=1
    if b**power3>x:
        power3-=1
    return power3
```

PIPELINE ANALYSIS

- Dataset: solutions collected from 6.00x (edX Fall '12)

Problem	Solutions	Represented by	Running Time
power	3875	862	15.5m
hangman	1118	552	8.1m
derivative	1433	1109	10.3m

- With Stacey Terman, pipeline adapted to "normalize" incorrect programs as well

HIGHLIGHTING DIFFERENCES

1538

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

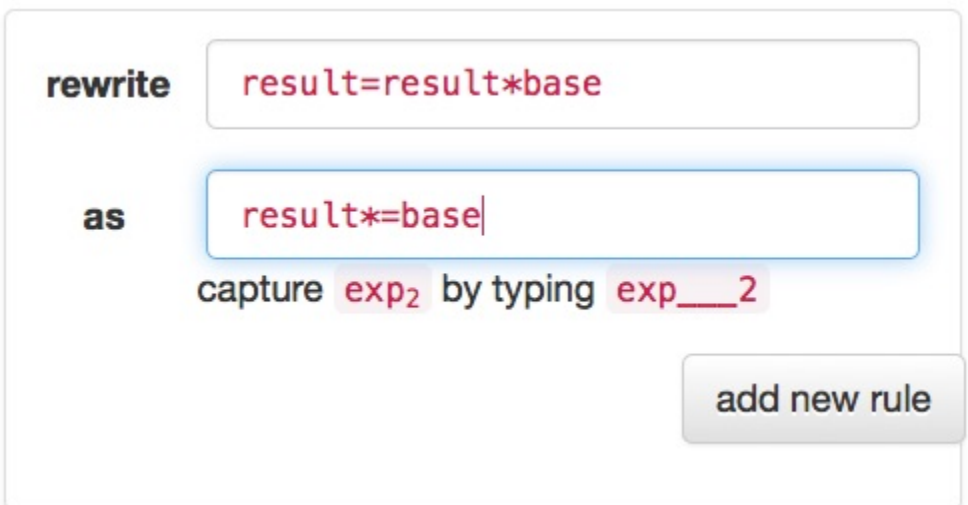
374

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

- Highlights differences in
  - solution popularity
  - syntax

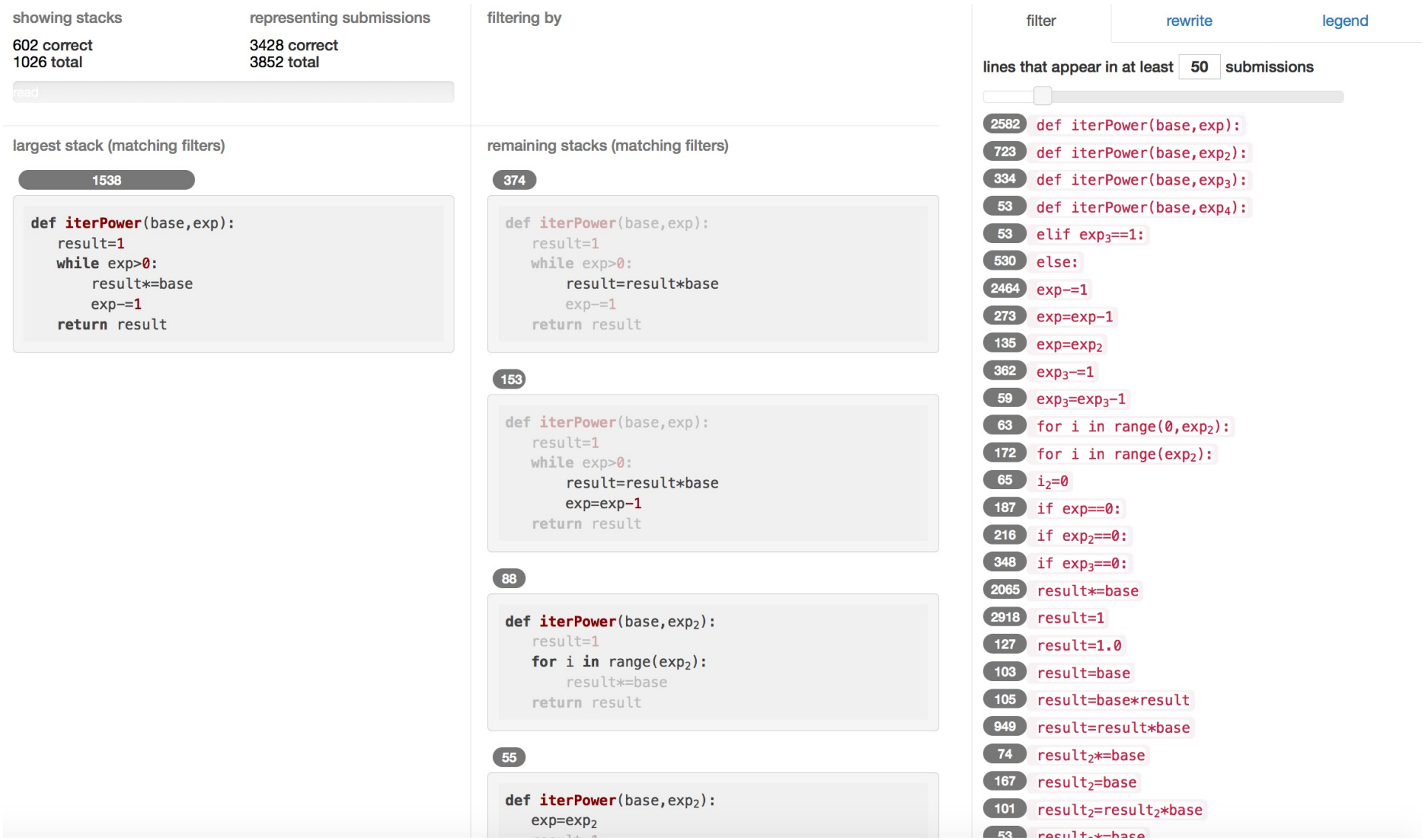
USERS CONTINUE SYNTHESIZING REPRESENTATIVES INTERACTIVELY

ADD NEW EQUIVALENCE RULES



OVERCODE

DEMO



(http://0.0.0.0:8000/?src=iterpower)

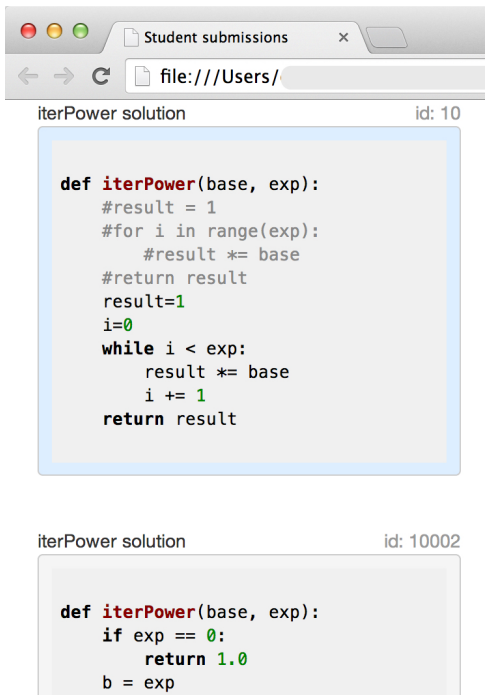
CAN TEACHERS READ AND UNDERSTAND MORE STUDENT CODE WITH OVERCODE?

- 12 participants (11 male)
- mean age: 25.4 ($\sigma = 6.9$)
- 4.9 ($\sigma = 3.0$) years of Python programming experience
- 9 previously graded code
- 5 previously graded Python code

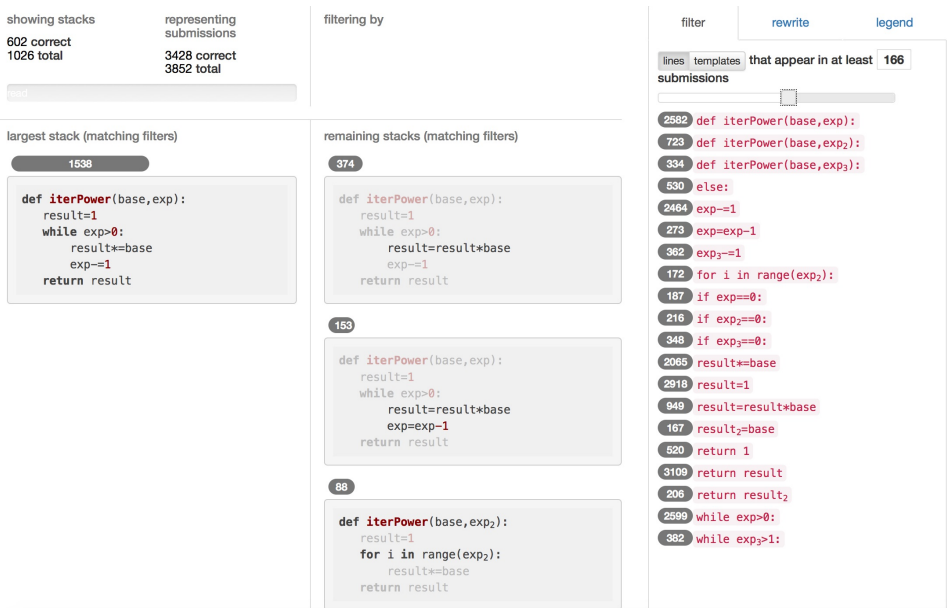
OVERCODE USER STUDY

- **Task**: Identify the 5 most frequent strategies used to solve the problem (15 min)
- For each strategy, they wrote:
  - a code example
  - comments
  - confidence

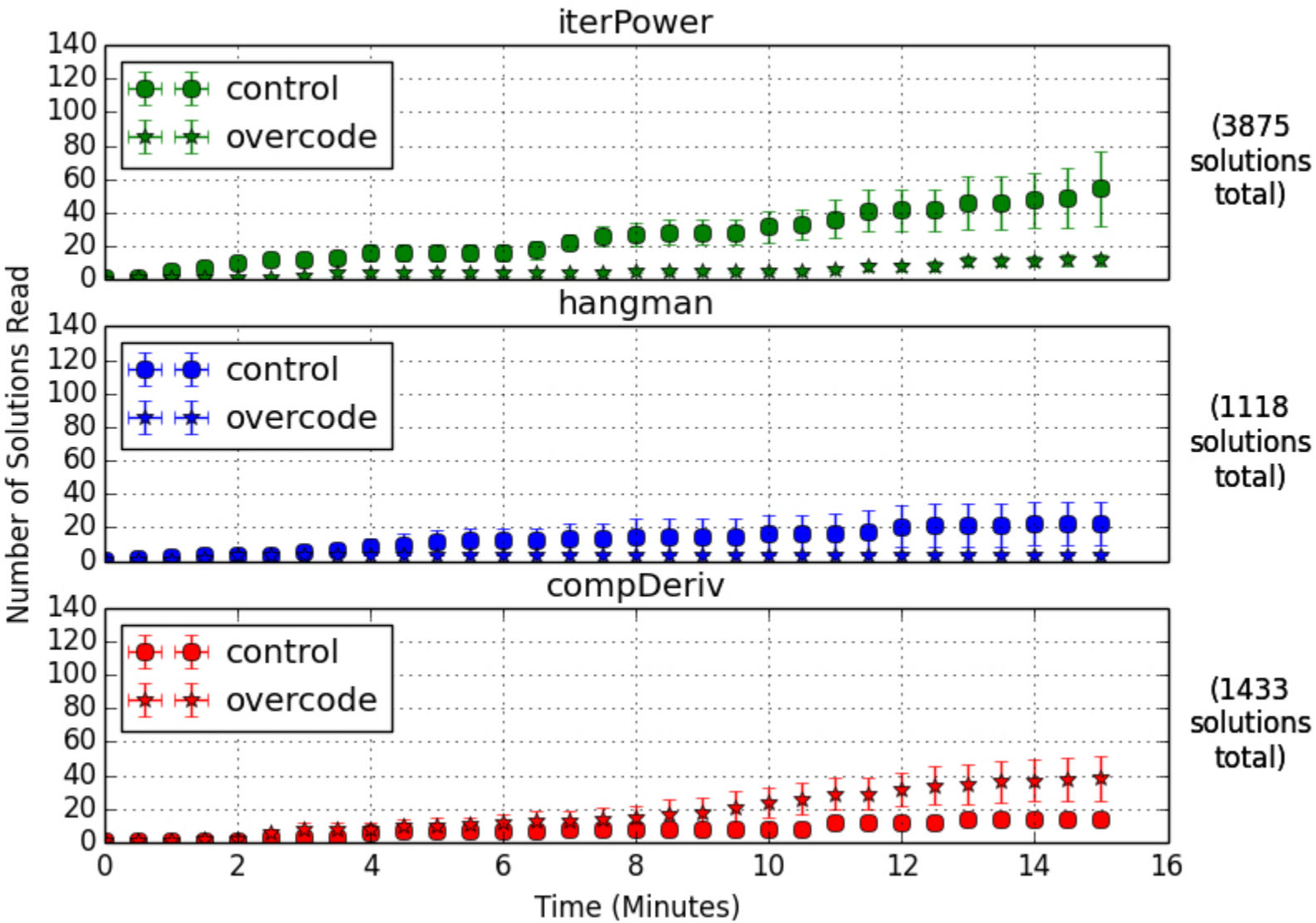
Control interface



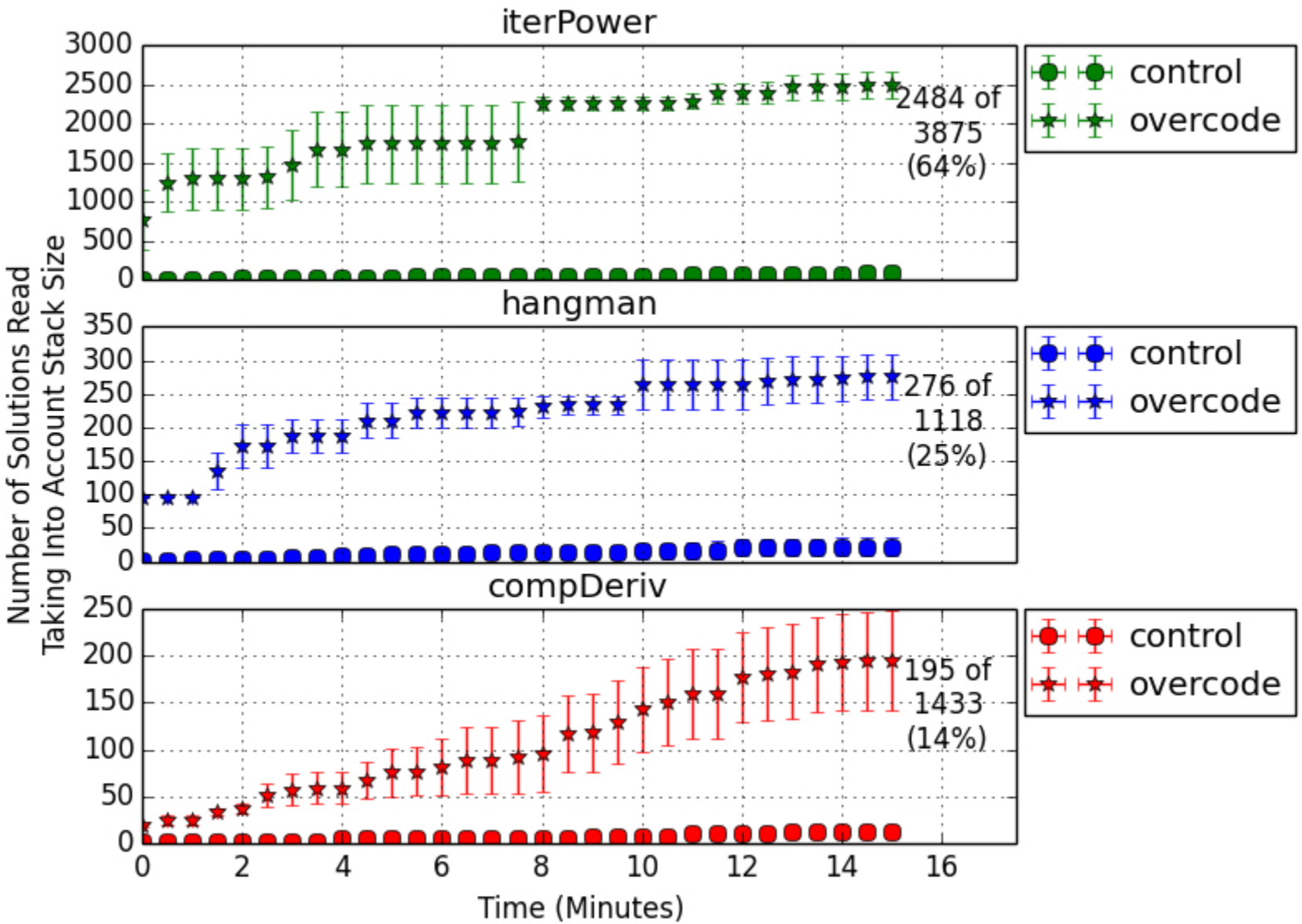
OverCode



SOLUTIONS READ

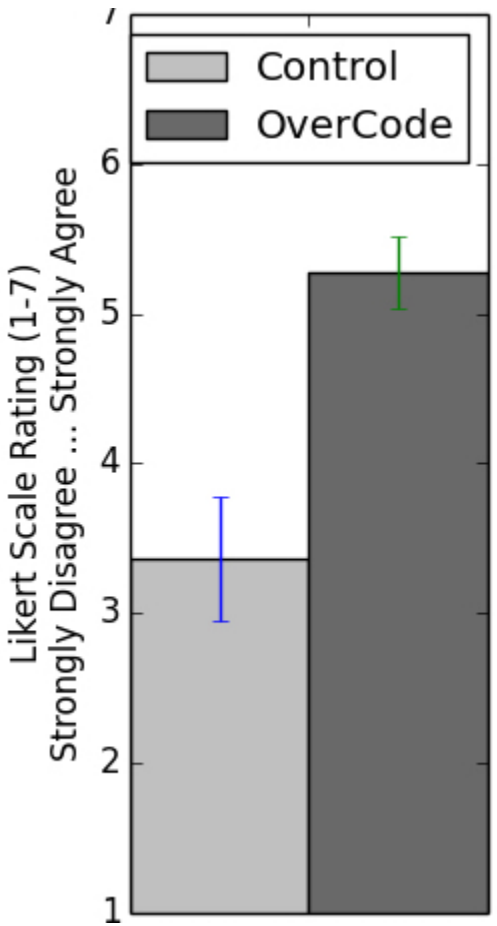


SOLUTIONS READ



PERCEIVED UNDERSTANDING

"The interface helped me develop a high-level view of students' understanding and misconceptions."



WHAT VARIATION DID WE IGNORE?

VARIABLE NAMING

Common Variable Name	Occur-rence Count	Sequence of Values	Original Variable Names
		iterPower	
result	3081	[1,5.0,25.0,125.0]	result, wynik, out, total, ans, acum, num, mult, output, ···
exp	2744	[3,2,1,0]	exp, iterator, app, ii, num, iterations, times, ctr, b, ···
exp	749	[3]	exp, count, temp, exp3, exp2, exp1, inexp, old_exp, ···
i	266	[0,1,2]	i, a, count, c, b, iterValue, iter, n, y, inc, x,times, ···
		hangman	
letter	817	[‘t’,‘i’,‘g’,‘e’,‘r’]	letter, char, item, i, letS, ch, c, lett, ···
result	291	[‘ ’,‘_’,‘_i’,‘_i_’,‘_i_e’,‘_i_e_’]	result, guessedWord, ans, str1, anss, guessed, string, ···
i	185	[0,1,2,3,4]	i, x, each, b, n, counter, idx, pos ···
found	76	[0,1,0,1,0]	found, n, letterGuessed, contains, k, checker, test, ···
		compDeriv	
result	1186	[[],[0.0],· · ·,[0.0,35.0,9.0,4.0]]	result, output, poly_deriv, res, deriv, resultPoly, ···
i	284	[-13.39,0.0,17.5,3.0,1.0]	i, each, a, elem, number, value, num, ···
i	261	[0,1,2,3,4,5]	i, power, index, cm, x, count, pwr, counter, ···
length	104	[5]	length, nmax, polyLen, lpoly, lenpoly, z, l, n, ···

EXPAND DIMENSION OF VARIATION

VARIABLE NAMES

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

1538 / 3853 solutions

tag	base	3846	tag	exp	2755	tag	result	3095
	base	3735		exp	2612		result	2495
	a	8		b	8		result	2495
	base	3735		exp	2612		ans	117
	base	3735		exp	2612		res	91
	base	3735		exp	2612		power	32
	base	3735		exp	2612		answer	31
	base	3735		exp	2612		x	28
	base	3735		exp	2612		r	21
	base	3735		exp	2612		a	20
	base	3735		exp	2612		value	17
	base	3735		exp	2612		total	17
	base	3735		exp	2612		resultado	13
	base	3735		exponente	1		resultado	13
	base	3735		exp	2612		ret	8
	base	3735		exp	2612		val	7
	base	3735		exp	2612		results	7
	base	3735		exp	2612		p	7
	base	3735		exp	2612		i	5
	base	3735		exp	2612		out	5

EXPAND DIMENSION OF VARIATION

VARIABLE NAMES

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

Synthesized representative solution

Common variables base, exp, and result

1538 / 3853 solutions

tag	base	3846	tag	exp	2755	tag	result	3095
	base	3735		exp	2612		result	2495
	a	8		b	8		result	2495
	base	3735		exp	2612		ans	117
	base	3735		exp	2612		res	91
	base	3735		exp	2612		power	32
	base	3735		exp	2612		answer	31
	base	3735		exp	2612		x	28
	base	3735		exp	2612		r	21
	base	3735		exp	2612		a	20
	base	3735		exp	2612		value	17
	base	3735		exp	2612		total	17
	base	3735		exp	2612		resultado	13
	base	3735		exponente	1		resultado	13
	base	3735		exp	2612		ret	8
	base	3735		exp	2612		val	7
	base	3735		exp	2612		results	7
	base	3735		exp	2612		p	7
	base	3735		exp	2612		i	5

EXPAND DIMENSION OF VARIATION

VARIABLE NAMES

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

Common variable result
(1, 5, 25, 125)

1538 / 3853 solutions

tag	base	3846	tag	exp	2755	tag	result	3095
base	3735		exp	2612		result	2495	
a	8		b	8		result	2495	
base	3735		exp	2612		ans	117	
base	3735		exp	2612		res	91	
base	3735		exp	2612		power	32	
base	3735		exp	2612		answer	31	
base	3735		exp	2612		x	28	
base	3735		exp	2612		r	21	
base	3735		exp	2612		a	20	
base	3735		exp	2612		value	17	
base	3735		exp	2612		total	17	
base	3735		exp	2612		resultado	13	
base	3735		exponente	1		resultado	13	
base	3735		exp	2612		ret	8	
base	3735		exp	2612		val	7	
base	3735		exp	2612		results	7	
base	3735		exp	2612		p	7	
base	3735		exp	2612		i	5	
base	3735		exp	2612		out	5	

EXPAND DIMENSION OF VARIATION

VARIABLE NAMES

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

Common variable result
(1, 5, 25, 125)

1538 / 3853 solutions

tag	base	3846	tag	exp	2755	tag	result	3095
	base	3735		exp	2612		result	2495
	a	8		b	8		result	2495
	base	3735		exp	2612		ans	117
	base	3735		exp	2612		res	91
	base	3735		exp	2612		power	32
	base	3735		exp	2612		answer	31
	base	3735		exp	2612		x	28
	base	3735		exp	2612		r	21
	base	3735		exp	2612		a	20
	base	3735		exp	2612		value	17
	base	3735		exp	2612		total	17
	base	3735		exp	2612		resultado	13
	base	3735		exponente	1		resultado	13
	base	3735		exp	2612		ret	8
	base	3735		exp	2612		val	7
	base	3735		exp	2612		results	7
	base	3735		exp	2612		p	7
	base	3735		exp	2612		i	5
	base	3735		exp	2612		out	5

EXPAND DIMENSION OF VARIATION

VARIABLE NAMES

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

Common variable result
(1, 5, 25, 125)

1538 / 3853 solutions

tag	base	3846	tag	exp	2755	tag	result	3095
	base	3735		exp	2612		result	2495
	a	8		b	8		result	2495
	base	3735		exp	2612		ans	117
	base	3735		exp	2612		res	91
	base	3735		exp	2612		power	32
	base	3735		exp	2612		answer	31
	base	3735		exp	2612		x	28
	base	3735		exp	2612		r	21
	base	3735		exp	2612		a	20
	base	3735		exp	2612		value	17
	base	3735		exp	2612		total	17
	base	3735		exp	2612		resultado	13
	base	3735		exponente	1		resultado	13
	base	3735		exp	2612		ret	8
	base	3735		exp	2612		val	7
	base	3735		exp	2612		results	7
	base	3735		exp	2612		p	7
	base	3735		exp	2612		i	5
	base	3735		exp	2612		out	5

EXPAND DIMENSION OF VARIATION

VARIABLE NAMES

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

Common variable result
(1, 5, 25, 125)

1538 / 3853 solutions

tag	base	3846	tag	exp	2755	tag	result	3095
	base	3735		exp	2612		result	2495
	a	8		b	8		result	2495
	base	3735		exp	2612		ans	117
	base	3735		exp	2612		res	91
	base	3735		exp	2612		power	32
	base	3735		exp	2612		answer	31
	base	3735		exp	2612		x	28
	base	3735		exp	2612		r	21
	base	3735		exp	2612		a	20
	base	3735		exp	2612		value	17
	base	3735		exp	2612		total	17
	base	3735		exp	2612		resultado	13
	base	3735		exponente	1		resultado	13
	base	3735		exp	2612		ret	8
	base	3735		exp	2612		val	7
	base	3735		exp	2612		results	7
	base	3735		exp	2612		p	7
	base	3735		exp	2612		i	5
	base	3735		exp	2612		out	5

FOOBAZ

(We'll come back to this system soon, to answer R3.)

(R1) UNDERSTAND SOLUTION VARIATION

- OverCode helps teachers
  - read more complicated solutions
  - "read" more solutions
  - Foobaz is an example of how to expand previously hidden dimensions of variation

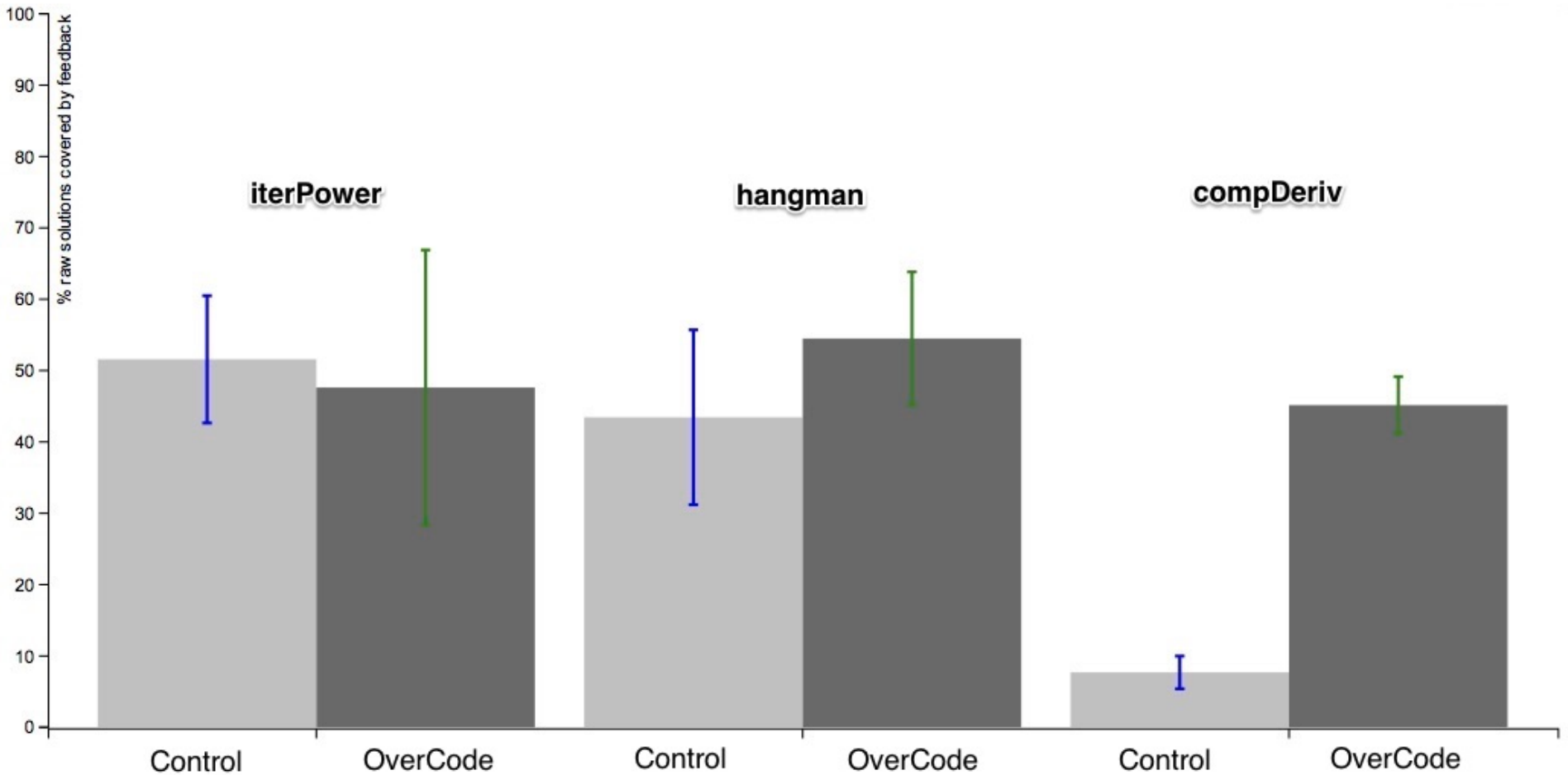
OUTLINE

- Solution Variation (SV)
- Research Questions
  - (R1) understand SV
  - ⇒ (R2) feedback on **SV** at scale
  - (R3) personalized feedback on **SV** at scale
  - (R4) collect and distribute student advice, given **SV**
- Discussion

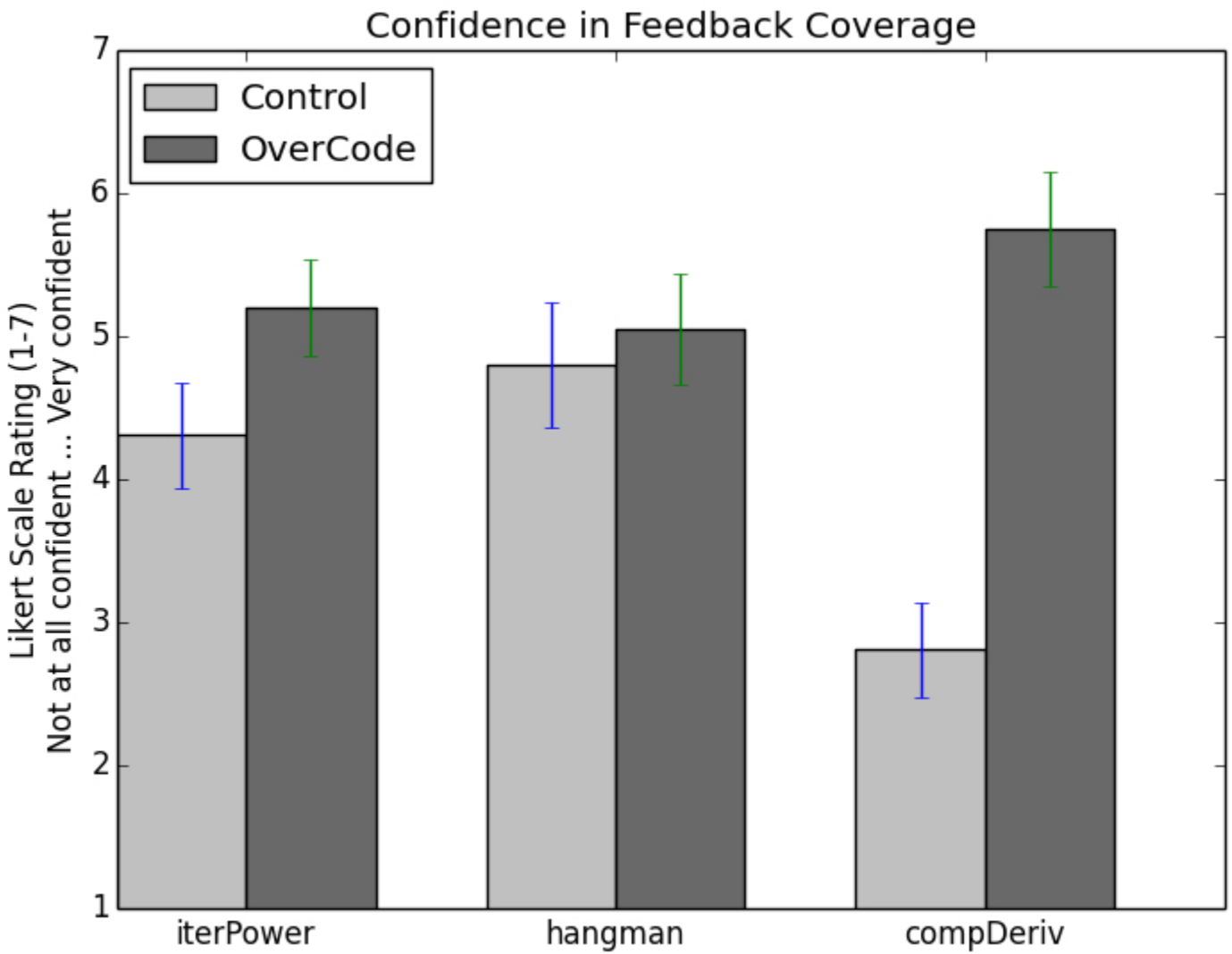
FEEDBACK ON APPROACH & READABILITY AT SCALE

- Same subjects and task as before
- Hypotheses for OverCode condition:
  - More feedback coverage
  - More perceived coverage

% OF RAW SOLUTIONS COVERED BY FEEDBACK



PERCEIVED FEEDBACK COVERAGE



(R2) FEEDBACK ON SOLUTION VARIATION AT SCALE

- OverCode helps teachers
  - write feedback that is relevant to more solutions

OUTLINE

- Solution Variation (SV)
- Research Questions
  - (R1) understand SV
  - (R2) feedback on SV at scale
  - ⇒ (R3) personalized feedback on **SV** at scale
  - (R4) collect and distribute student advice, given **SV**
- Discussion

WHY PERSONALIZE FEEDBACK?

- One-on-one tutoring is a gold standard *(Bloom '84)*
- Good tutors *(Lepper and Wolverton's INSPIRE model)*
  - Ask students to explain their thinking *(Chi '94)*
  - Progressively reveal content
    - Keep students with their zone of proximal development *(Lev Vygotsky)*

PERSONALIZED FEEDBACK ON READABILITY

VARIABLE NAMES

IN THE FORM OF PERSONALIZED ACTIVE LEARNING EXERCISES ("QUIZ") ON GOOD AND BAD NAMES

WHY VARIABLE NAMES?

- Most basic form of documentation
- Without modifying execution
  - express type and purpose of an object
  - suggest kinds of operators that manipulate it
- Some intro students mix themselves up through naming (Guttag)

```
for i in range(len(inputArray)):
    a += myFunction(i)
```

`i` is OK (idiomatic)

`a` is not OK (too short)

```
for i in inputArray:
    result += i
```

`i` is not OK (misleading)

`result` is OK

QUOTE ABOUT VARIABLE NAMES

- Donald Knuth compares a good programmer to an essayist, who:

“with thesaurus in hand, chooses the names of variables carefully and explains what each variable means”

HOW DO WE GIVE PERSONALIZE FEEDBACK ON VARIABLE NAMES AT SCALE?

PERSONALIZED FEEDBACK WORKFLOW

1. Teacher annotates good and bad variable names in student code (using Foobaz interface)
2. Teacher curates quizzes based on these examples
3. Foobaz analyzes student solutions
4. Foobaz delivers personalized quizzes for each student solution

PERSONALIZED FEEDBACK WORKFLOW

- (Step 1) Teacher annotates a few good and bad examples of variable names for several **common variables**

Common Variable Name	Occur-rence Count	Sequence of Values	Original Variable Names
			iterPower
result	3081	[1,5.0,25.0,125.0]	result, wynik, out, total, ans, acum, num, mul, output, ···
exp	2744	[3,2,1,0]	exp, iterator, pow, ii, num, iterations, time, ctr, b, ···
exp	749	[3]	exp, count, temp, exp3, exp, exp1, inexp, old_exp, ···
i	266	[0,1,2]	i, a, count, c, b, iterValue, iter, n, y, inc, x,times, ···

PERSONALIZED FEEDBACK WORKFLOW

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

Teacher Annotations

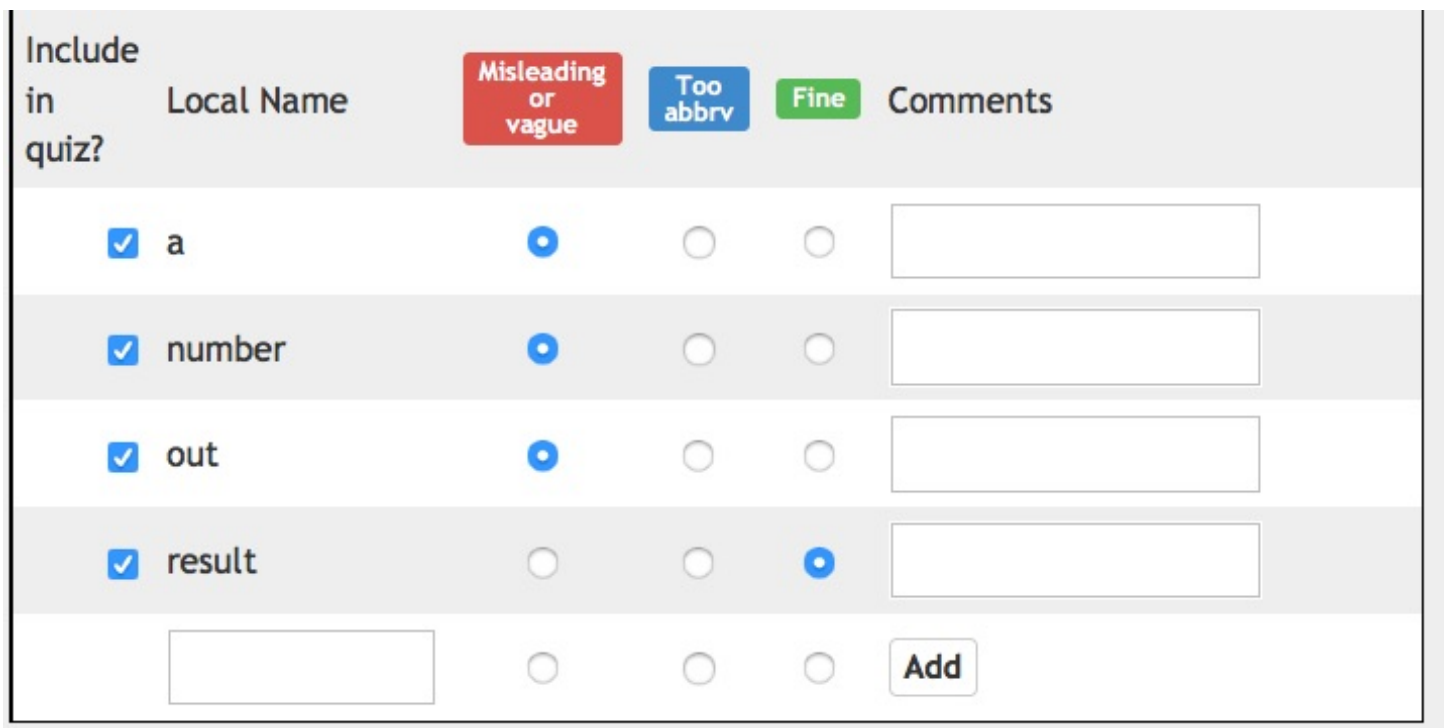
1538 / 3853 solutions

tag	base	3646	tag	exp	2755	tag	result	3095
	base	3735		exp	2612	Fine	result	2495
Misleading or vague	a	8	Misleading or vague	b	8	Fine	result	2495
	base	3735		exp	2612		ans	117
	base	3735		exp	2612		res	91
	base	3735		exp	2612		power	32
	base	3735		exp	2612		answer	31
	base	3735		exp	2612		x	28
	base	3735		exp	2612		r	21
	base	3735		exp	2612	Too Short	a	20
	base	3735		exp	2612		value	17
	base	3735		exp	2612		total	17
	base	3735		exp	2612		resultado	13
	base	3735		exponente	1		resultado	13

PERSONALIZED FEEDBACK WORKFLOW

- (Step 2) Teacher adds commentary, adds or removes choices

Example: quiz template for **common variable** `result`



PERSONALIZED FEEDBACK WORKFLOW

- Example student solution:

```
def iterPower(base,exp):
  myAns=1
  for j in range(exp):
      myAns*=base
  return myAns
```

- (Step 3) Foobaz identifies common variables
  - `myAns` is an instance of **common variable** `result`
  - `j` is an instance of **common variable** `i`

PERSONALIZED FEEDBACK WORKFLOW

- (Step 4) Foobaz delivers personalized variable name quiz

You recently wrote the following solution, and we've replaced one variable's name with **A**:

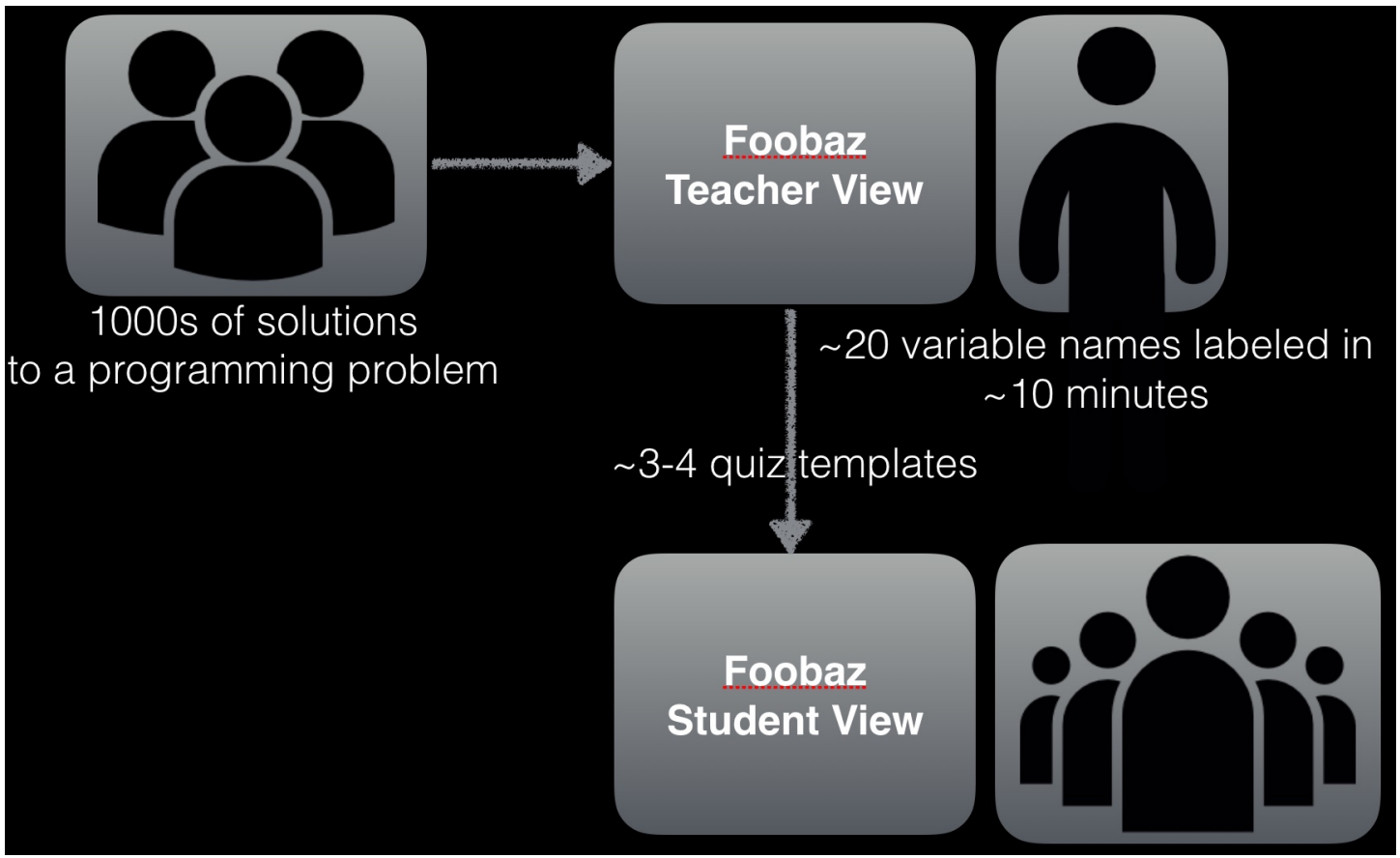
```
def iterPower(base,exp):
    A=1
    for j in range(exp):
        A*=base
    return A
```

Rate the quality of the following names for the bold symbol A:

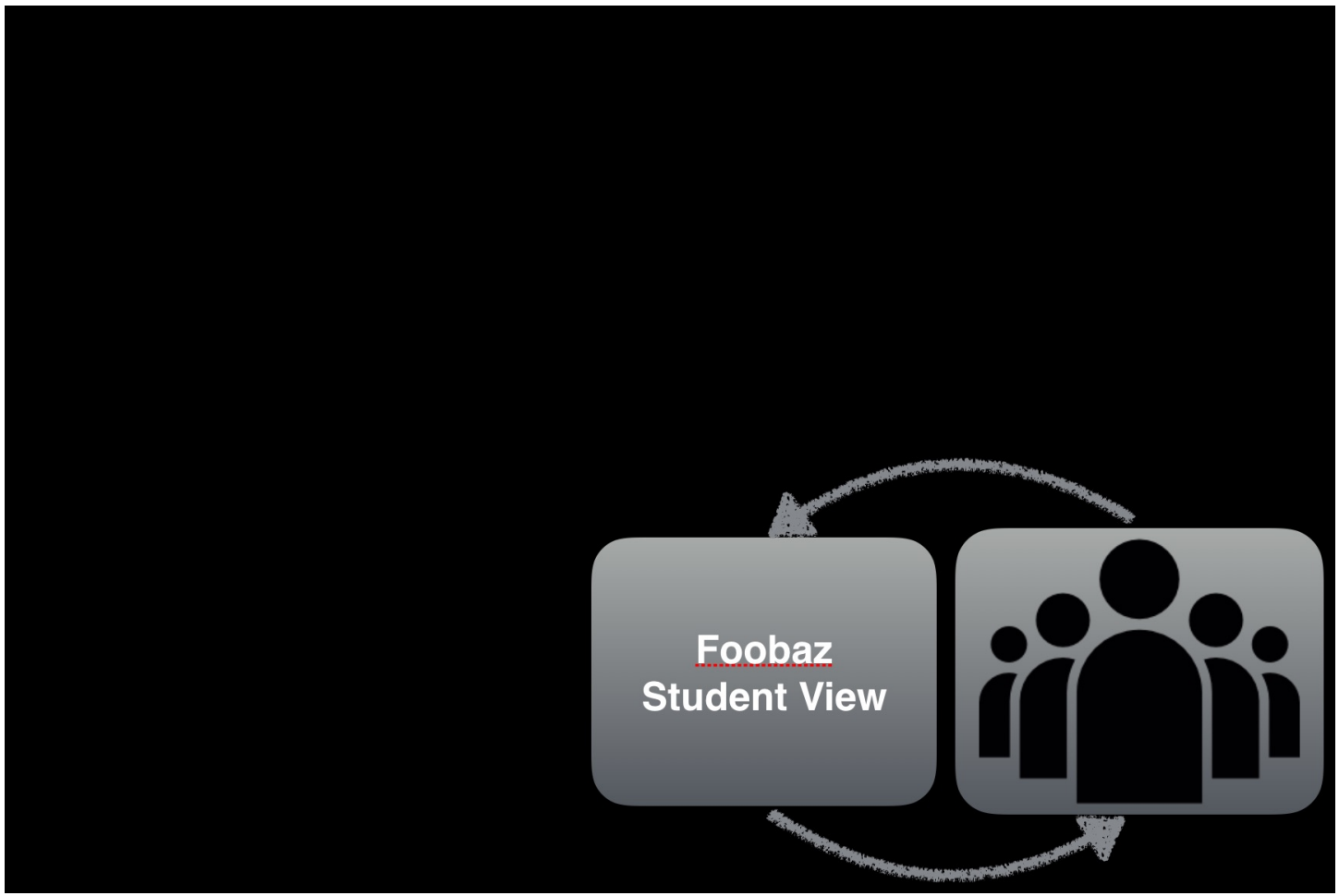
Name	Misleading or vague	Too abbrv	Fine
a	○	○	○
number	○	○	○
out	○	○	○
result	○	○	○

Compare to teacher

PERSONALIZED FEEDBACK WORKFLOW



PERSONALIZED FEEDBACK WORKFLOW



USER STUDY FOR TEACHERS

- Participants: 10 teachers
  - Python-course TAs/LAs/graders
  - (6 female, mean age: 23)
- Same datasets as before (`iterPower`, `hangman`, `derivative`)
- Task: Generate personalizable quizzes for as many students as possible

USER STUDY FOR TEACHERS

RESULTS

- In 4-11 minutes, each teacher
  - labeled an average of 21 out of 670-929 unique variable names
  - covered at least 75% of student solutions with at least one personalized variable name quiz

(R3) PERSONALIZED FEEDBACK ON SOLUTION VARIATION AT SCALE

- Foobaz workflow helps teachers scale up one-on-one conversations about variable name choices
  - conversations → active learning exercises (quizzes)

OUTLINE

- Solution Variation (SV)
- Research Questions
  - (R1) understand SV
  - (R2) feedback on SV at scale
  - (R3) personalized feedback on SV at scale
  - ⇒ (R4) collect and distribute student advice, given **SV**
- Discussion

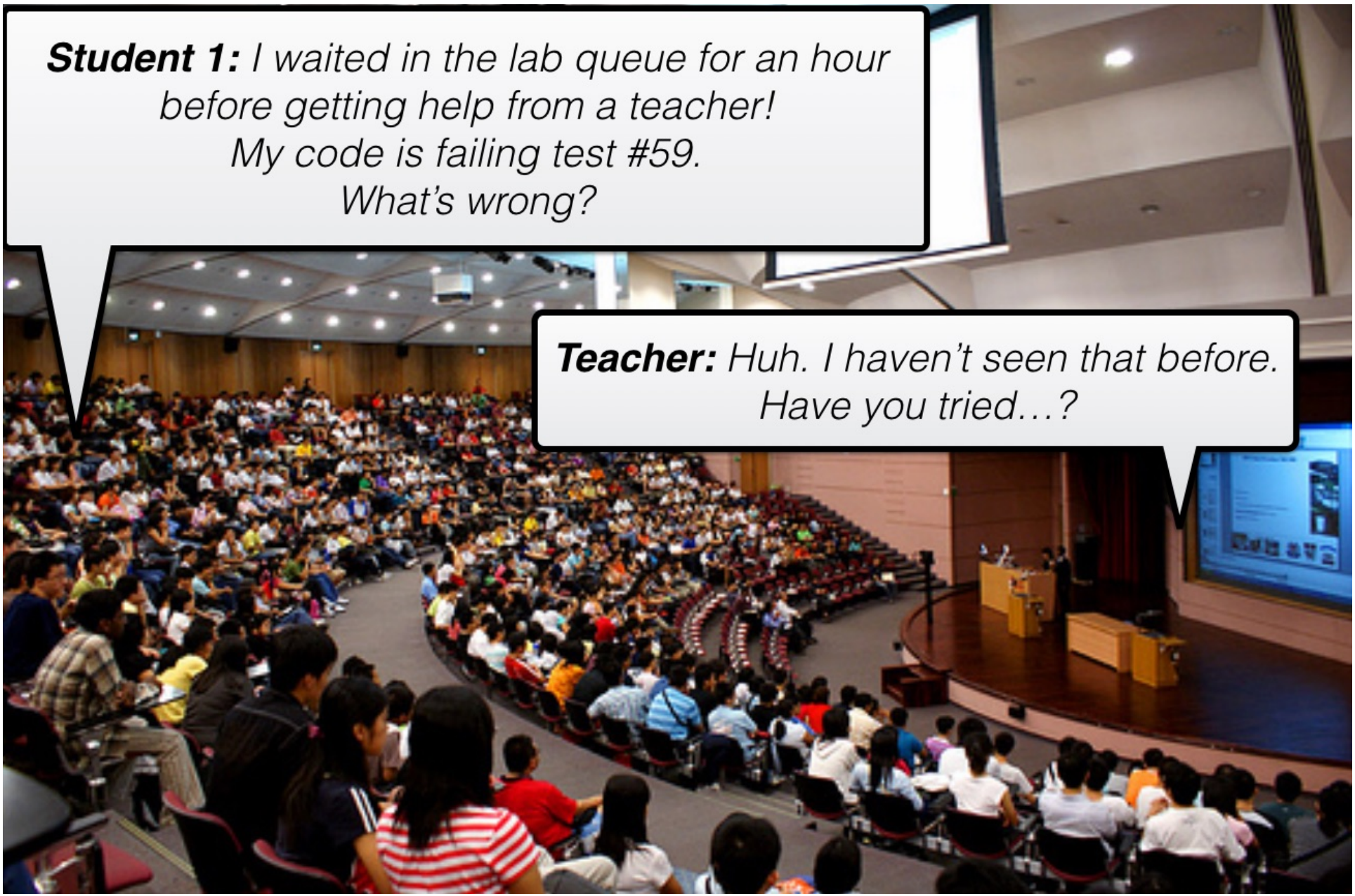
LEARNERSOURCING PERSONALIZED HINTS

- **Problem**: it's hard to get personalized help in large classes
  - especially when there are many solutions (and bugs)
- **Key Insight**: Students who struggle, then succeed, become experts on particular solutions (and bugs)
- **Approach**: Collect and distribute hints by students who earned the expertise necessary to write them

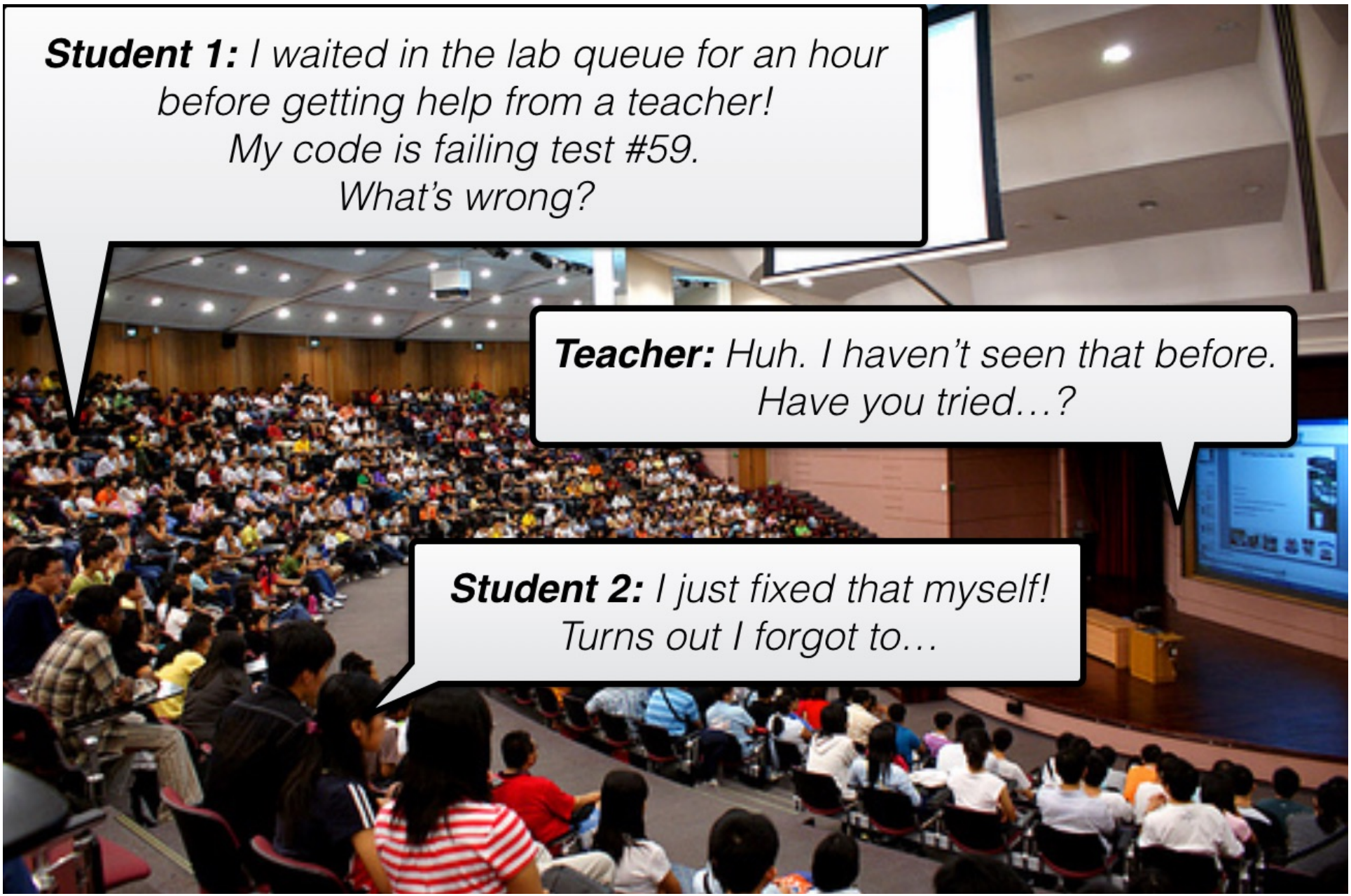
LEARNERSOURCING

- **Prior Art**: *Learnersourcing* puts learners to work while they learn (Kim ’13)
- This is *targeted learnersourcing*: hint writers and receivers are not drawn from pool of all students

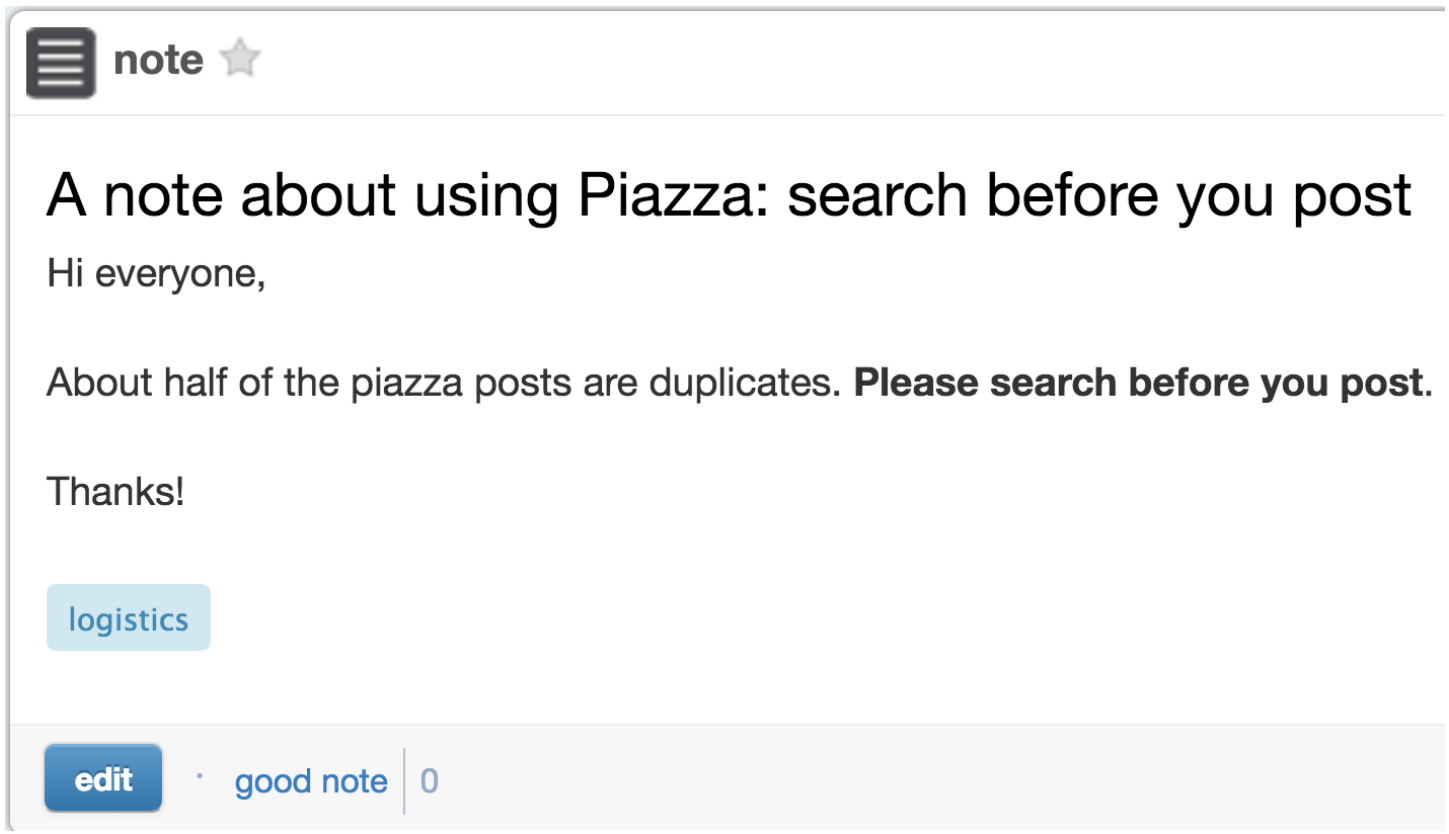
DEBUGGING


Student 1: I waited in the lab queue for an hour before getting help from a teacher! My code is failing test #59. What's wrong?
Teacher: Huh. I haven't seen that before. Have you tried…?

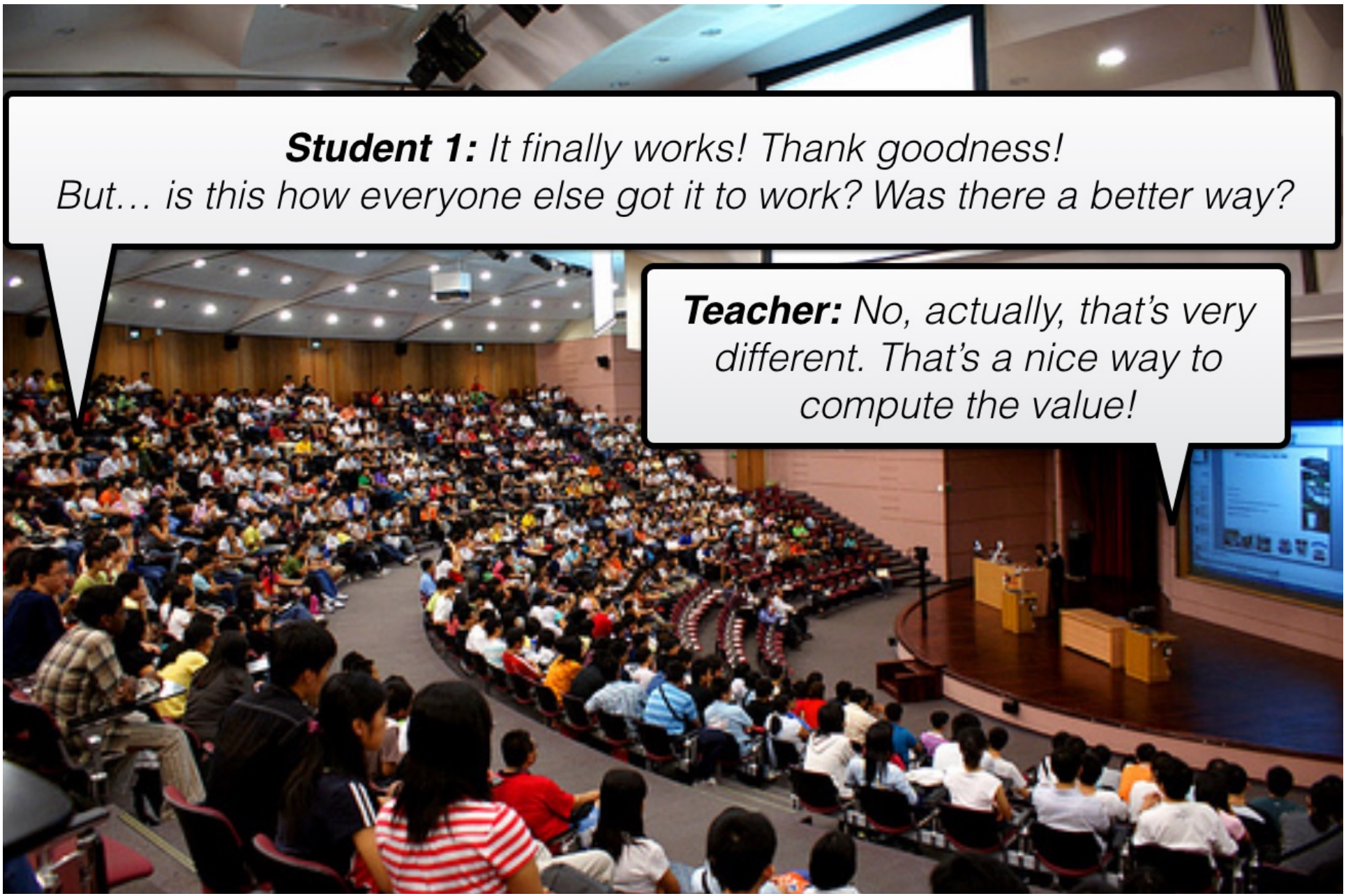
DEBUGGING


Student 1: I waited in the lab queue for an hour before getting help from a teacher! My code is failing test #59. What's wrong?
Teacher: Huh. I haven't seen that before. Have you tried…?
Student 2: I just fixed that myself! Turns out I forgot to…

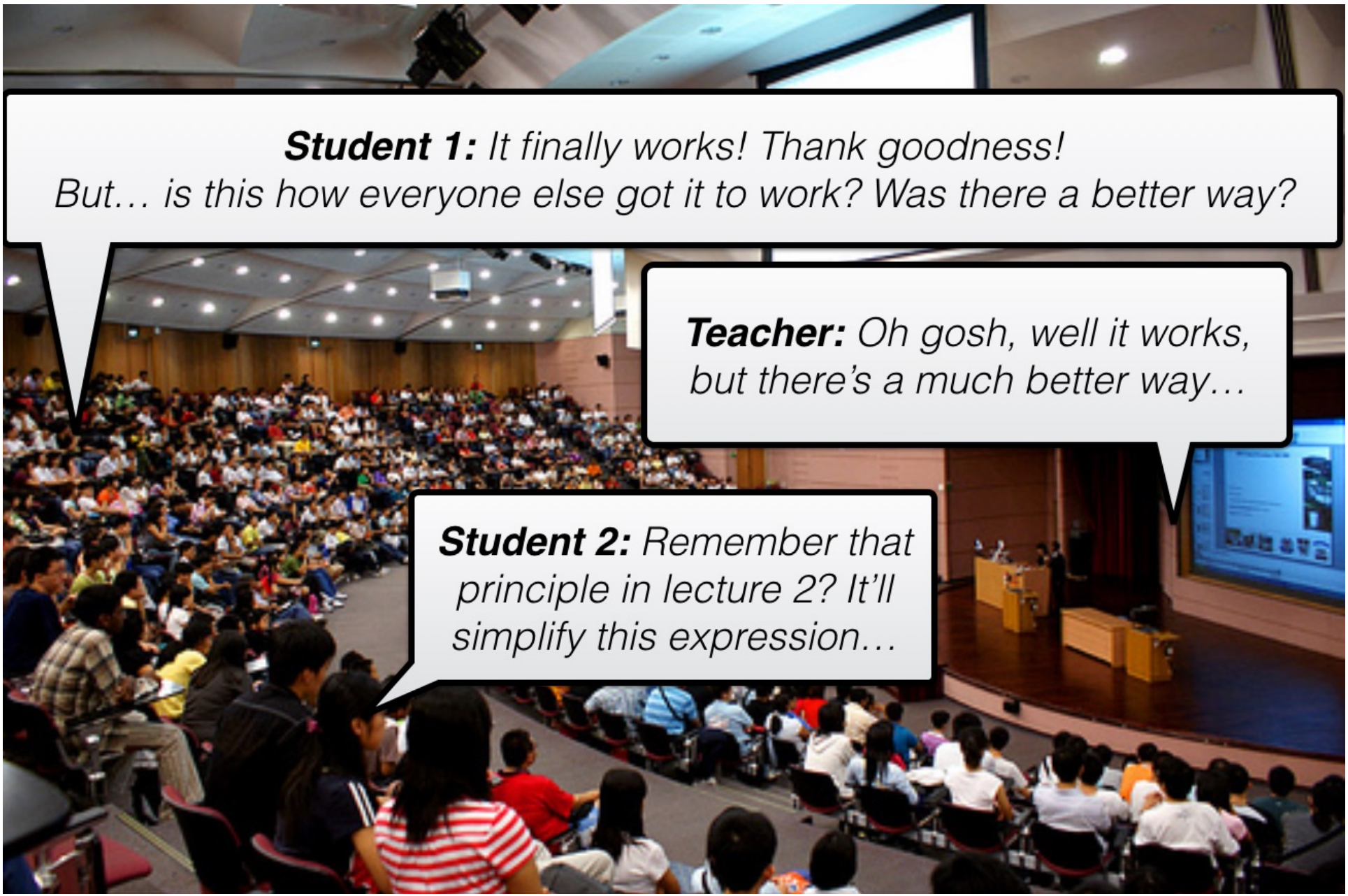
STUDENTS DON'T ALWAYS DESCRIBE THEIR PROBLEMS IN AN EASILY SEARCHABLE WAY



RECOGNIZING GOOD SOLUTIONS



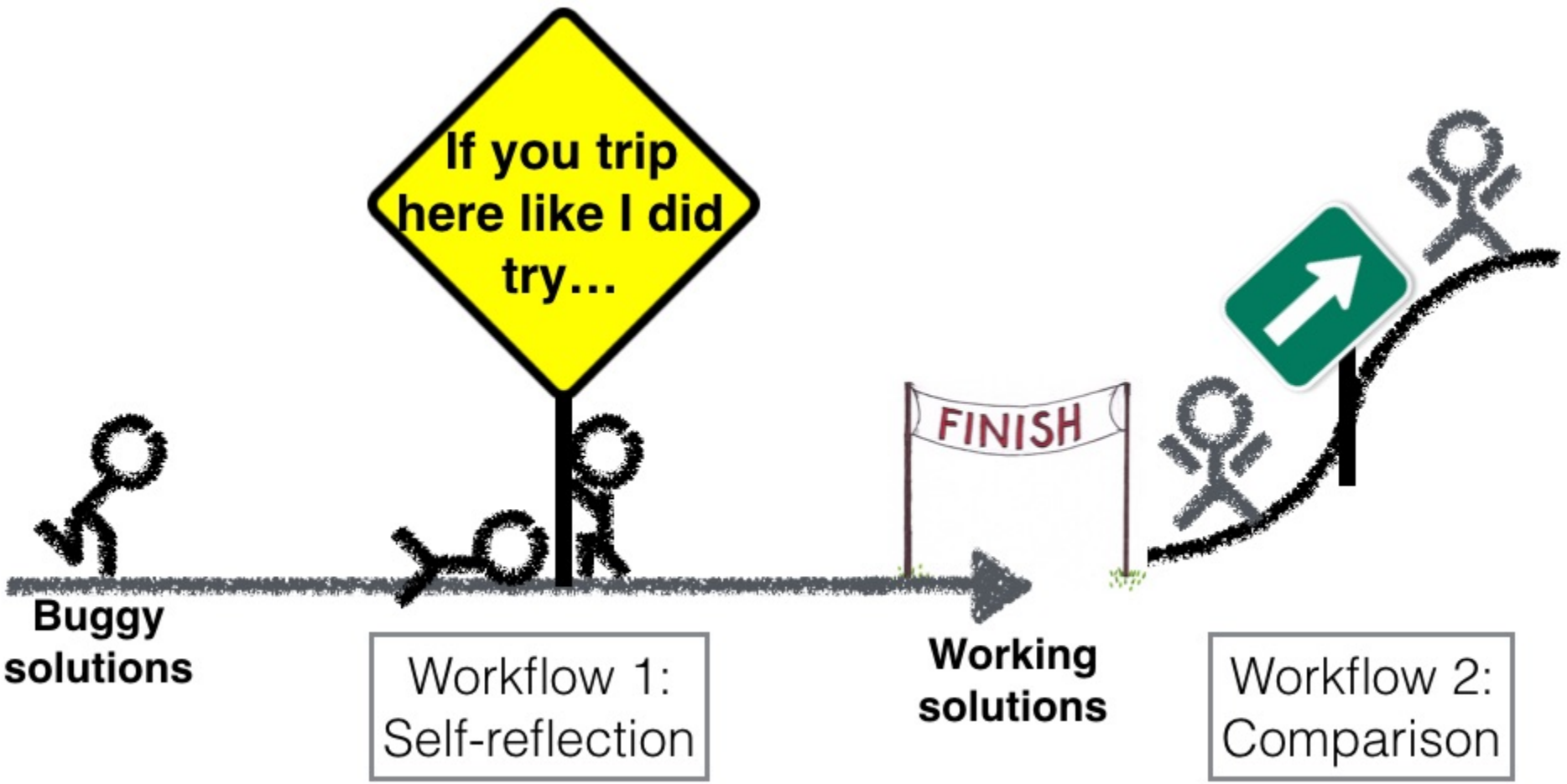
IMPROVING SOLUTIONS



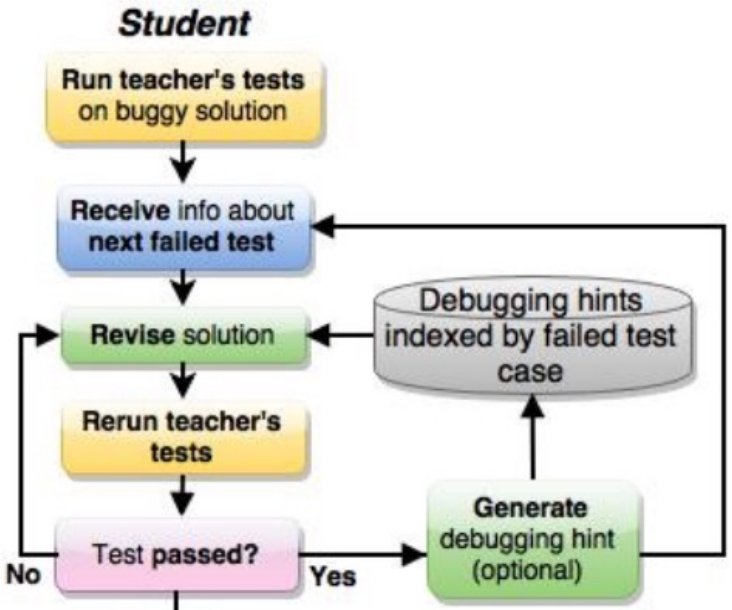
INSIGHTS

- **Debugging**: Students who fix bug_x can offer hints for students still stuck on bug_x
- **Solution Improvment**: Students can
  - compare their solutions to other student solutions
  - write hints to help others improve their own solutions

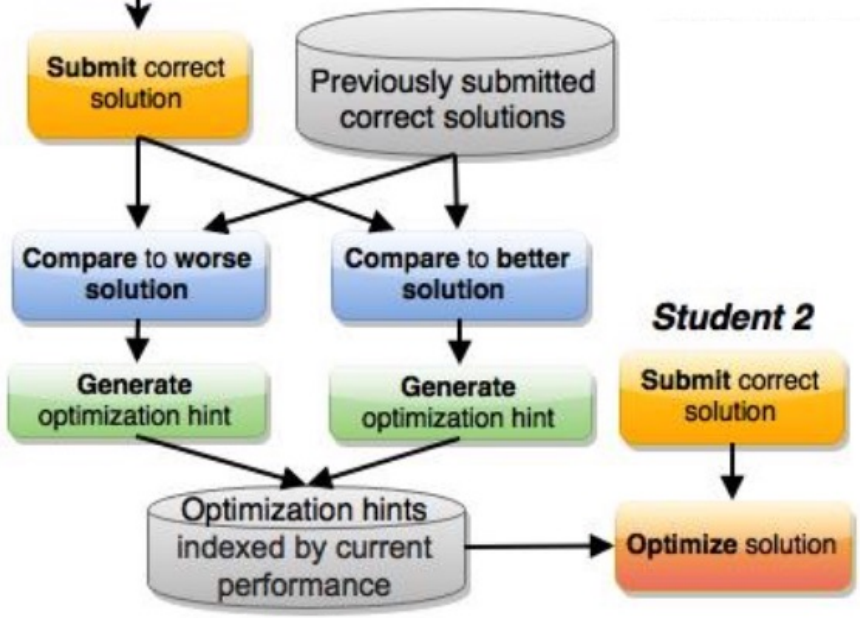
CONTRIBUTION: TWO TARGETED LEARNERSOURCING WORKFLOWS



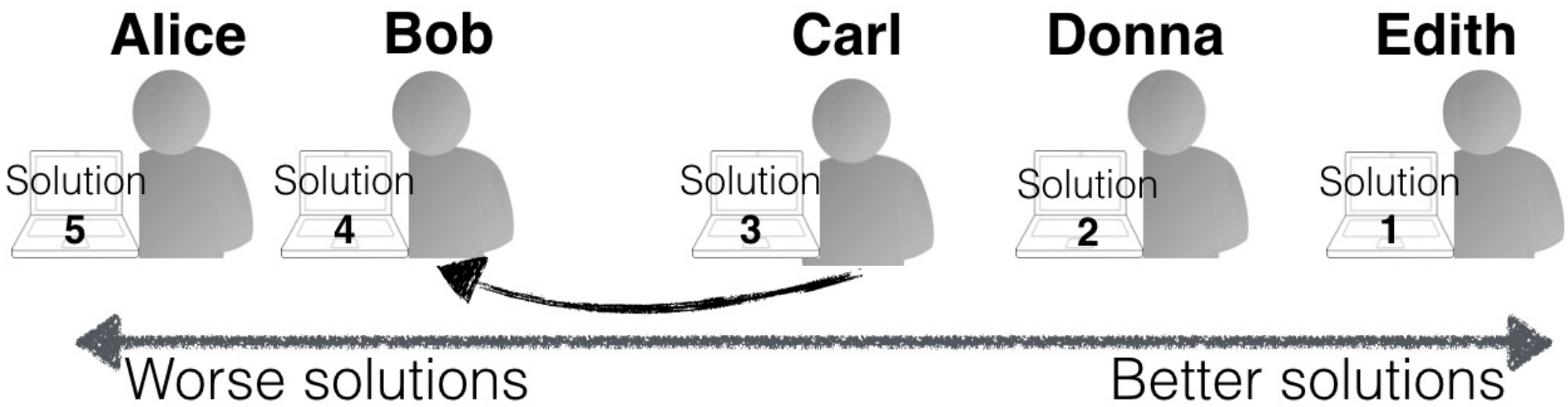
Workflow 1: Self-reflection

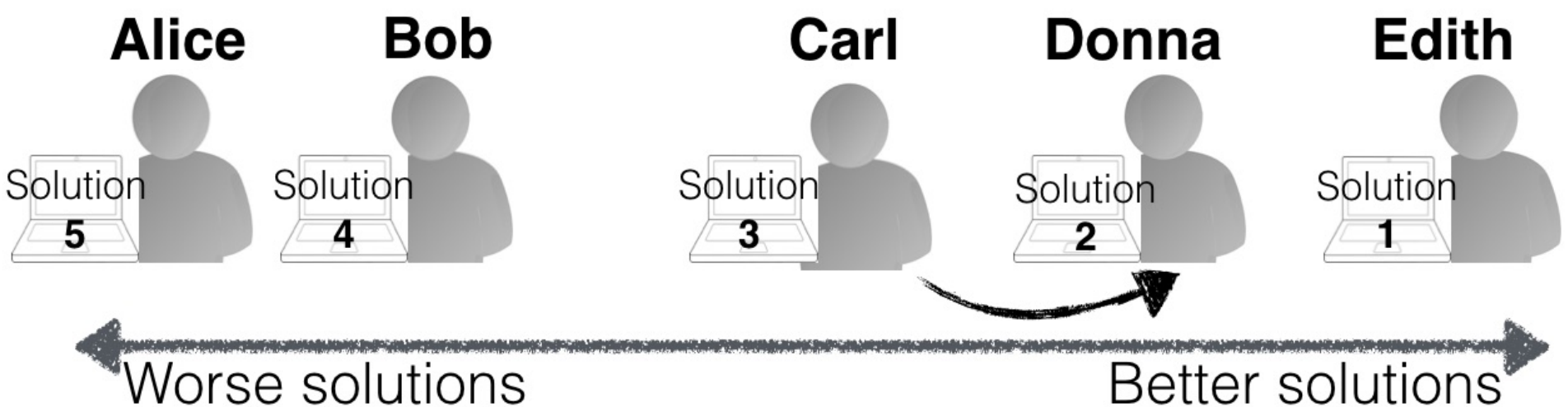


Workflow 2: Comparison



COMPARISON WORKFLOW





EVALUATION IN MIT HARDWARE DESIGN CLASS (>200 STUDENTS)

- Self-reflection workflow
  - deployed over multiple semesters
  - active engagment by students
  - most successful when endorsed by staff
- Comparison workflow
  - solution comparison & hint collection added to lab
  - gave student-written hints to 9 students in 1 hr lab study
  - hints helped students improve solutions

(R4) COLLECT AND DISTRIBUTE STUDENT ADVICE, GIVEN SOLUTION VARIATION

- Demonstrated two targeted learnersourcing workflows that work when there are many different solutions (and bugs)
- **Benefits**:
  - Opportunities for student reflection & peer teaching
  - Awareness of alternative solutions
  - Reduced dependence on staff

OUTLINE

- Solution Variation (SV)
- Research Questions
  - (R1) understand SV
  - (R2) feedback on SV at scale
  - (R3) personalized feedback on SV at scale
  - (R4) collect and distribute student advice, given SV
- Discussion

THESIS STATEMENT

- *Clustering and visualizing solution variation* collected from programming courses can help teachers
  - gain insights into student design choices
  - enhance test cases
  - give personalized style feedback at scale
  - collect and distribute student-written hints

- Thesis Contributions: systems and workflows that demonstrate these benefits

RELATED WORK

- Mining patterns in code
  - Huang et al.'s "CodeWebs" @ Stanford (WWW '14)
  - Fast et al.'s "Codex" @ Stanford (CHI '14)
- Mining design choices
  - Kumar et al.'s "Webzeitgeist" (CHI '13)
- Mining variable names
  - Høst and Østvold's data-driven *Programmer English*
- Teacher-writen style feedback
  - AutoStyle @ UCBerkeley (L@S '15, ITS '16)
- Prototype-based clustering of solutions
  - Gross et al. (DeLFI '12)

CLUSTERING REPRESENTATIVE SOLUTIONS

Interactive Bayesian Case Model Empowering Humans via Intuitive Interaction

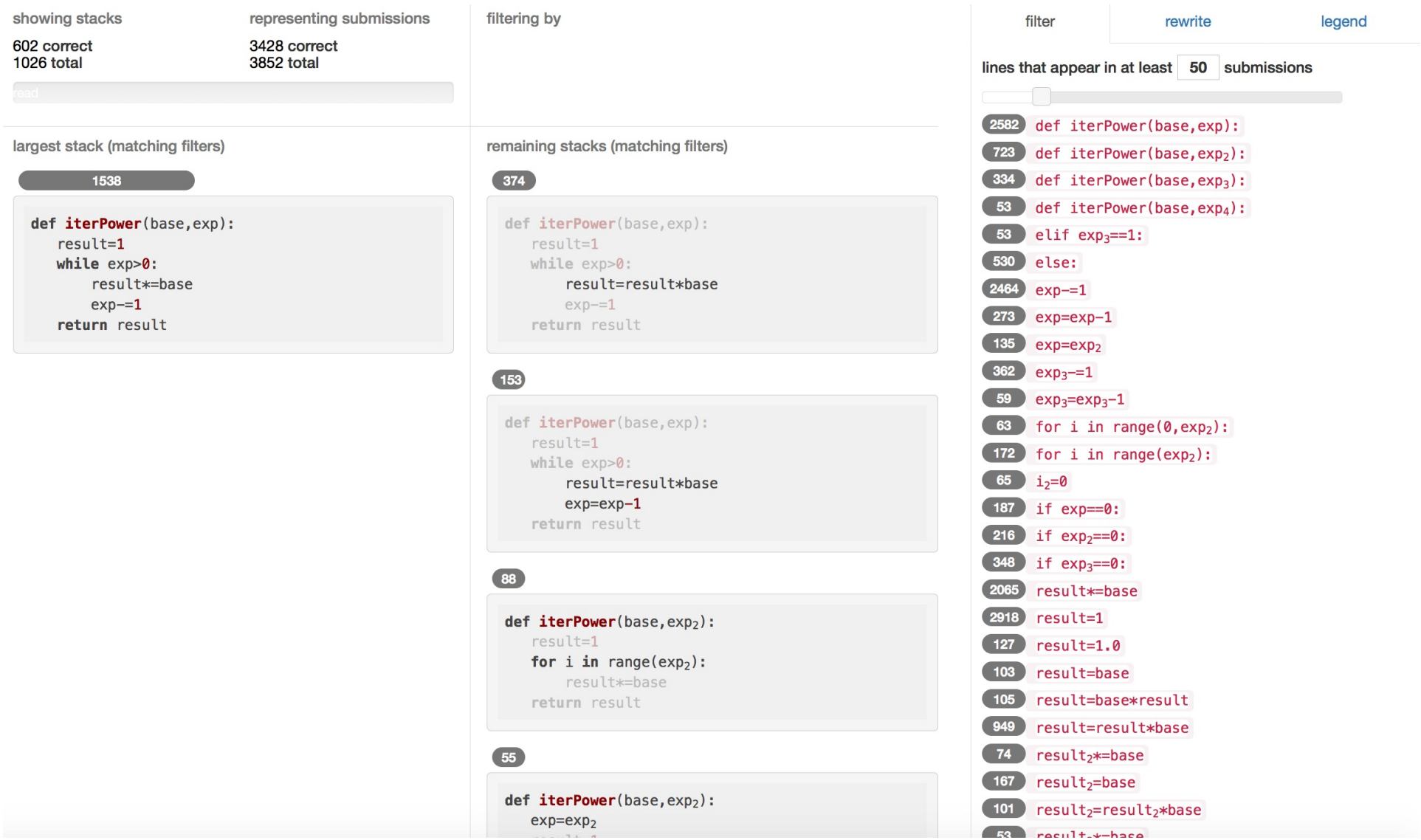
ACKNOWLEDGEMENTS

- MIT CSAIL
- UID group
- Massachusetts Wrestling Community
- Family & Friends

SUMMARY

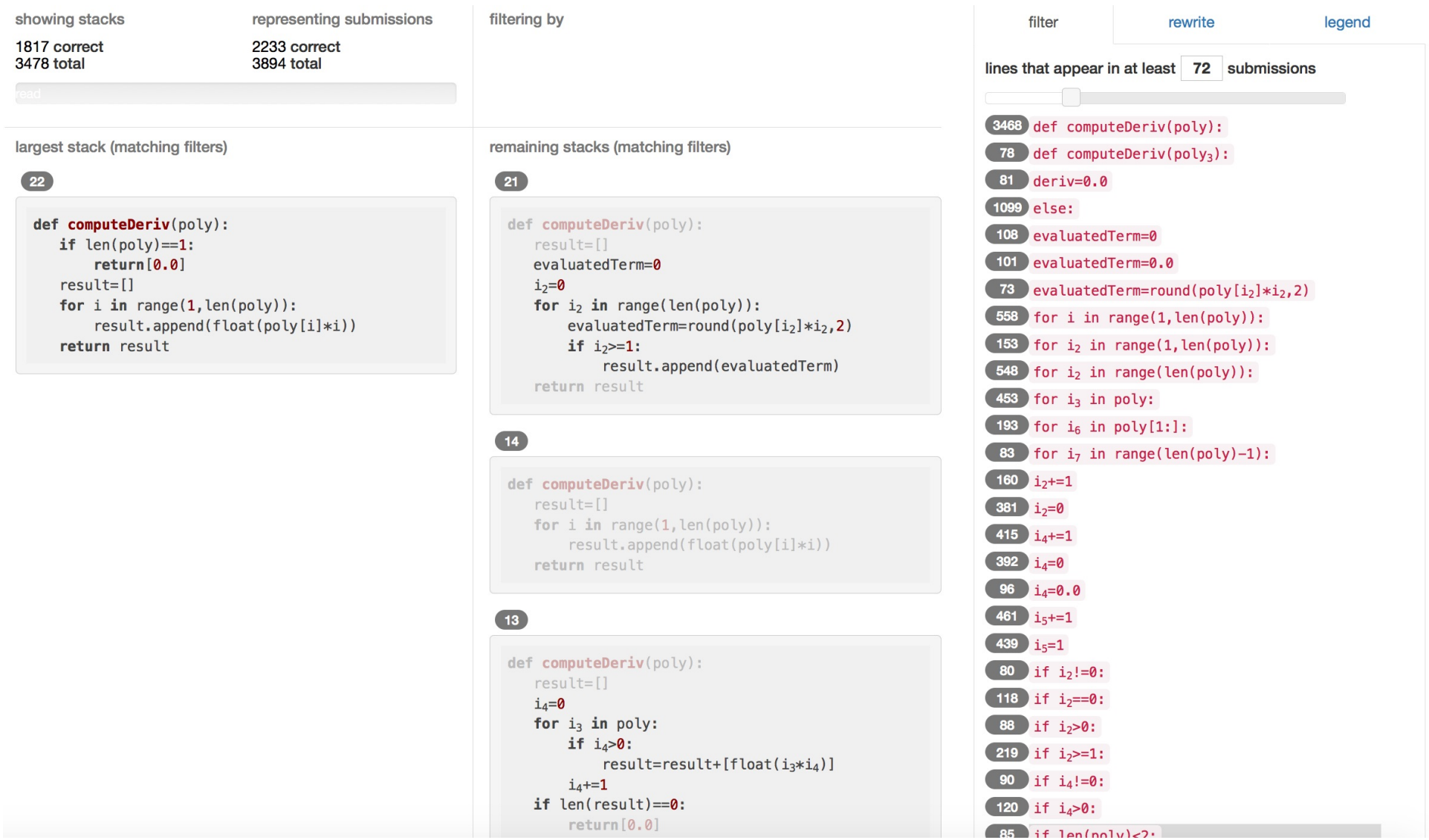
- *Clustering and visualizing solution variation* collected from programming courses can help teachers
  - gain insights into student design choices
  - enhance test cases
  - give personalized style feedback at scale
  - collect and distribute student-written hints

OVERCODE



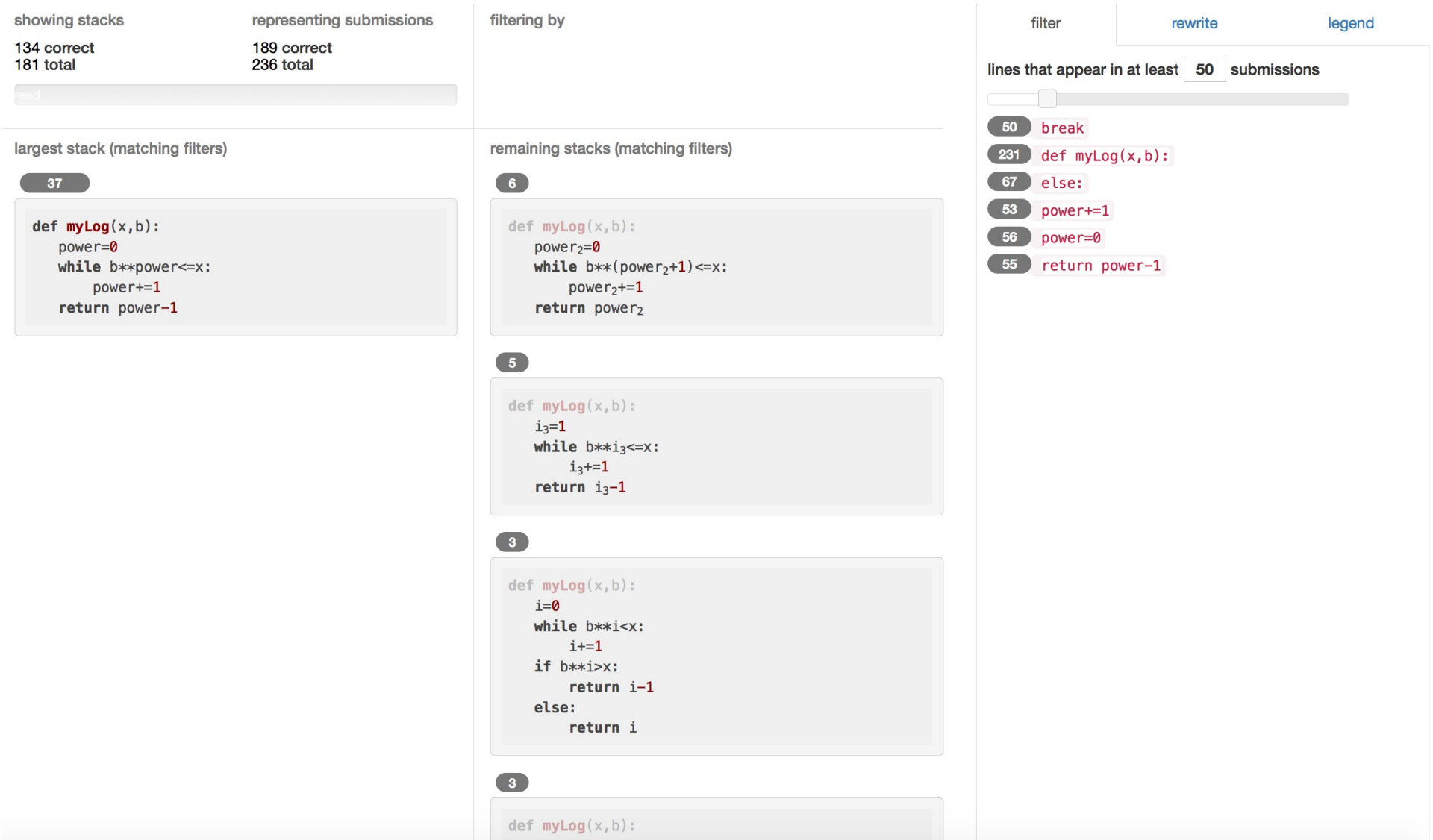
(http://0.0.0.0:8000/?src=iterpower)

OVERCODE



(http://0.0.0.0:8000/?src=compderiv)

OVERCODE



(http://0.0.0.0:8000/?src=mylog)

OVERCODE

showing stacks
84 correct
115 total

representing submissions
160 correct
191 total

filtering by

filter | rewrite | legend

lines that appear in at least 50 submissions

84 d_copy=d.copy()
74 d_copy[key]+=1
72 d_copy[key]-=1
168 def give_and_take(d,L):
162 else:
146 if key in L:
50 new_dict={}
103 return d_copy

largest stack (matching filters)

15

```
def give_and_take(d,L):
    new_dict={}
    for key in d:
        if key in L:
            new_dict[key]=d[key]+1
        else:
            new_dict[key]=d[key]-1
    return new_dict
```

remaining stacks (matching filters)

15

```
def give_and_take(d,L):
    d_copy=d.copy()
    for key in d_copy.keys():
        if key in L:
            d_copy[key]+=1
        else:
            d_copy[key]-=1
    return d_copy
```

12

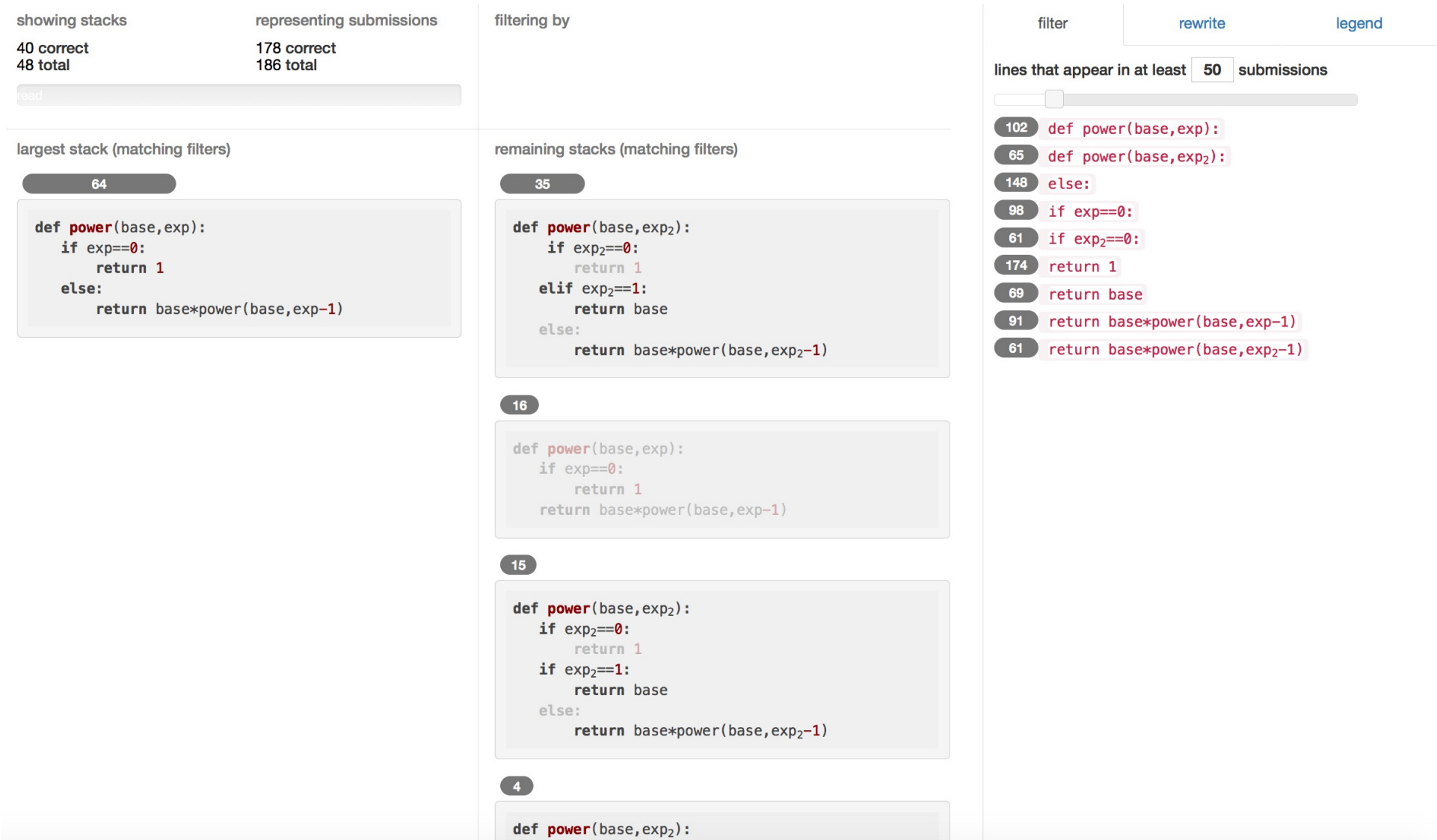
```
def give_and_take(d,L):
    new_dict={}
    for key in d.keys():
        if key in L:
            new_dict[key]=d[key]+1
        else:
            new_dict[key]=d[key]-1
    return new_dict
```

11

```
def give_and_take(d,L):
    d_copy=d.copy()
    for key in d_copy:
        if key in L:
            d_copy[key]+=1
        else:
            d_copy[key]-=1
```

(http://0.0.0.0:8000/?src=giveandtake)

OVERCODE



(http://0.0.0.0:8000/?src=q4_power)

OVERCODE

showing stacks
47 correct
121 total

representing submissions
96 correct
170 total

filtering by

filter | rewrite | legend

lines that appear in at least 22 submissions

35 L.reverse()
41 L_2.reverse()
43 def deep_reverse(L):
75 def deep_reverse(L_2):
25 for i in L:
31 for i_3 in L_2:
27 i.reverse()
32 i_3.reverse()

largest stack (matching filters)

19

```
def deep_reverse(L):
    for i in L:
        i.reverse()
    L.reverse()
```

remaining stacks (matching filters)

15

```
def deep_reverse(L2):
    L2.reverse()
    for i3 in L2:
        i3.reverse()
```

6

```
def deep_reverse(L2):
    L2.reverse()
    for i2 in range(len(L2)):
        L2[i2].reverse()
```

5

```
def deep_reverse(L2):
    L2.reverse()
    for i3 in L2:
        i3.reverse()
    return L2
```

4

```
def deep_reverse(L):
    for i in L:
        i.reverse()
    L.reverse()
    return L
```

(http://0.0.0.0:8000/?src=q4_deep)