

RunEx: Augmenting Regular-Expression Code Search with Runtime Values

Ashley Ge Zhang ¹, Yan Chen ², Steve Oney ¹

¹University of Michigan

²Virginia Tech



Large Introductory Programming Courses



Large Introductory Programming Courses



Large Introductory Programming Courses

More than hundreds of
students

Large Introductory Programming Courses

```
: "bake", "beam", "confide", "gri  
e'):  
ense=past_tense+[n+'d']  
  
ense=past_tense+[n+'ed']
```

```
words = ["adopt", "bake", "beam", "confide", "gri  
past_tense = list()  
for word in words:  
    if word[-1] == "e":  
        word = word + "d"  
        past_tense.append(word)  
    else:  
        word = word + "ed"  
        past_tense.append(word)  
print(past_tense)
```

```
: "bake", "beam", "confide", "gri  
st()  
  
rds:  
| == "e":  
word + "d"  
ense.append(word)  
  
word + "ed"  
ense.append(word)
```

```
1  
  
words = ["adopt", "bake", "beam", "confide", "gri  
past=""  
for word in words:  
    last_letter=word[len(word)-1]  
    if last_letter=='e':  
        word=word+"d"  
    else:  
        word=word+"ed"  
    past=past+" "+word  
past_tense=past.split(" ")[1:]  
print(past_tense)
```

More than hundreds of students

Viewing students' solutions is challenging

Background

Large Introductory Programming Courses

The figure shows a grid of 40 code editor windows, each displaying a different Python script. The scripts are attempting to generate past tense forms from a list of words. Many scripts contain errors such as SyntaxError, AttributeError, and TypeError. The code varies in complexity, with some using simple loops and others employing more advanced techniques like list comprehensions or string manipulation.

More than hundreds of students

Viewing students' solutions is challenging

Large Introductory Programming Courses

```
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = []

for word in words:
    if word[-1] == "e":
        past_tense.append(word+"d")
    else:
        past_tense.append(word+"ed")
```

12

```
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = []
for word in words:
    if word[-1] == "e":
        word = word + "d"
    else:
        word = word + "ed"
past_tense.append(word)
```

```
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = []

for word in words:
    if word[-1] == 'e':
        past_tense.append(word + 'd')
    else:
        past_tense.append(word + 'ed')
```

1

```
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense=[]
for i in words :
    n=len(i)
    if i[n-1] == 'e':
        past_tense+=i+"e"
    else :
        past_tense+=i+"ed"
print(past_tense)
```

Not pass unit test

11

```
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = []
for wrd in words:
    if wrd[-1] == 'e':
        past_tense += [wrd+'d']
```

11

```
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = []

for word in words:
    if word[-1] == "e":
```

More than hundreds of students

Viewing students' solutions is challenging

Take significant effort to understand students' solutions

Large Introductory Programming Courses

1

```
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = []
for word in words:
    newWord = word
    if(word[-1] == "e"):
        newWord += "d"
    else:
        newWord += "ed"
    past_tense.append(newWord)
```

1

```
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = []
for word in words:
    newWord = word
    if(word[-1] == "e"):
        newWord += "d"
    else:
        newWord += "ed"
    past_tense.append(newWord)

print(words)
print(past_tense)
```

1

```
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = []
for word in words:
    newWord = word
    if(word[-1] == "e"):
        newWord += "d"
    else:
        newWord += "ed"
    past_tense.append(newWord)

print(words)
print(past_tense)
```

1

```
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = list()

for a in words:
    if a[-1] == 'e':
        old_time = a + 'd'
    else:
        old_time = a + 'ed'
    past_tense.append(old_time)

print(past_tense)
```

More than hundreds of students

Viewing students' solutions is challenging

Take significant effort to understand students' solutions

Understand Students' Learning Needs

For each word in **words**, add 'd' to the end of the word if the word ends in 'e' to make it past tense. Otherwise, add 'ed' to make it past tense. Save these past tense words to a list called **past_tense**.

Understand Students' Learning Needs

For each word in **words**, add 'd' to the end of the word if the word ends in 'e' to make it past tense. Otherwise, add 'ed' to make it past tense. Save these past tense words to a list called **past_tense**.

Problem Steps

1. Loop over **words**

Understand Students' Learning Needs

For each word in **words**, add 'd' to the end of the word if the word ends in 'e' to make it past tense. Otherwise, add 'ed' to make it past tense. Save these past tense words to a list called **past_tense**.

Problem Steps

1. Loop over **words**
2. Conditional statements

Understand Students' Learning Needs

For each word in **words**, add 'd' to the end of the word if the word ends in 'e' to make it past tense. Otherwise, add 'ed' to make it past tense. Save these past tense words to a list called **past_tense**.

Problem Steps

1. Loop over **words**
2. Conditional statements
3. Modify str variable

Understand Students' Learning Needs

For each word in **words**, add 'd' to the end of the word if the word ends in 'e' to make it past tense. Otherwise, add 'ed' to make it past tense. Save these past tense words to a list called **past_tense**.

Problem Steps

1. Loop over **words**
2. Conditional statements
3. Modify str variable
4. Append word to a list

Instructors' Needs

Problem Steps

1. Loop over **words**

2. Conditional statements

3. Modify str variable

4. Append word to a list

Questions Instructors want to know

Instructors' Needs

Problem Steps

1. Loop over **words**

2. Conditional statements

3. Modify str variable

4. Append word to a list

Questions Instructors want to know

- How many students correctly write a loop?

```
for i in words:
```

...

```
for i in enumerate(words):
```

Correct
90 students

Incorrect
10 students

```
# other mistakes  
...
```

Incorrect
30 students

Instructors' Needs

Problem Steps

1. Loop over **words**

2. Conditional statements

3. Modify str variable

4. Append word to a list

Questions Instructors want to know

- How many students correctly write a loop?
- How many students use an anti-pattern that works but goes against the principles being taught?
- ...

for key in dict:

...

For key in dict.keys():

...

Pattern taught in class

An anti-pattern that works

Instructors' Needs

Questions Instructors want to know

- How many students correctly write a loop?
- How many students use an anti-pattern that works but goes against the principles being taught?
- ...

Instructors' Needs

Instructors' Needs

Questions Instructors want to know

- How many students correctly write a loop?
- How many students use an anti-pattern that works but goes against the principles being taught?
- ...

Instructors' Needs

1. Find **common patterns and mistakes** in students' code

```
for i in words:  
    ...
```

```
if word.endswith('e'):  
    ...  
else:  
    ...
```

```
past_tense.append(word+'d' )
```

```
for i in enumerate(words):
```

```
if 'e' in word:
```

```
if word[len(word)] == 'e':
```

```
word.append('ed')
```

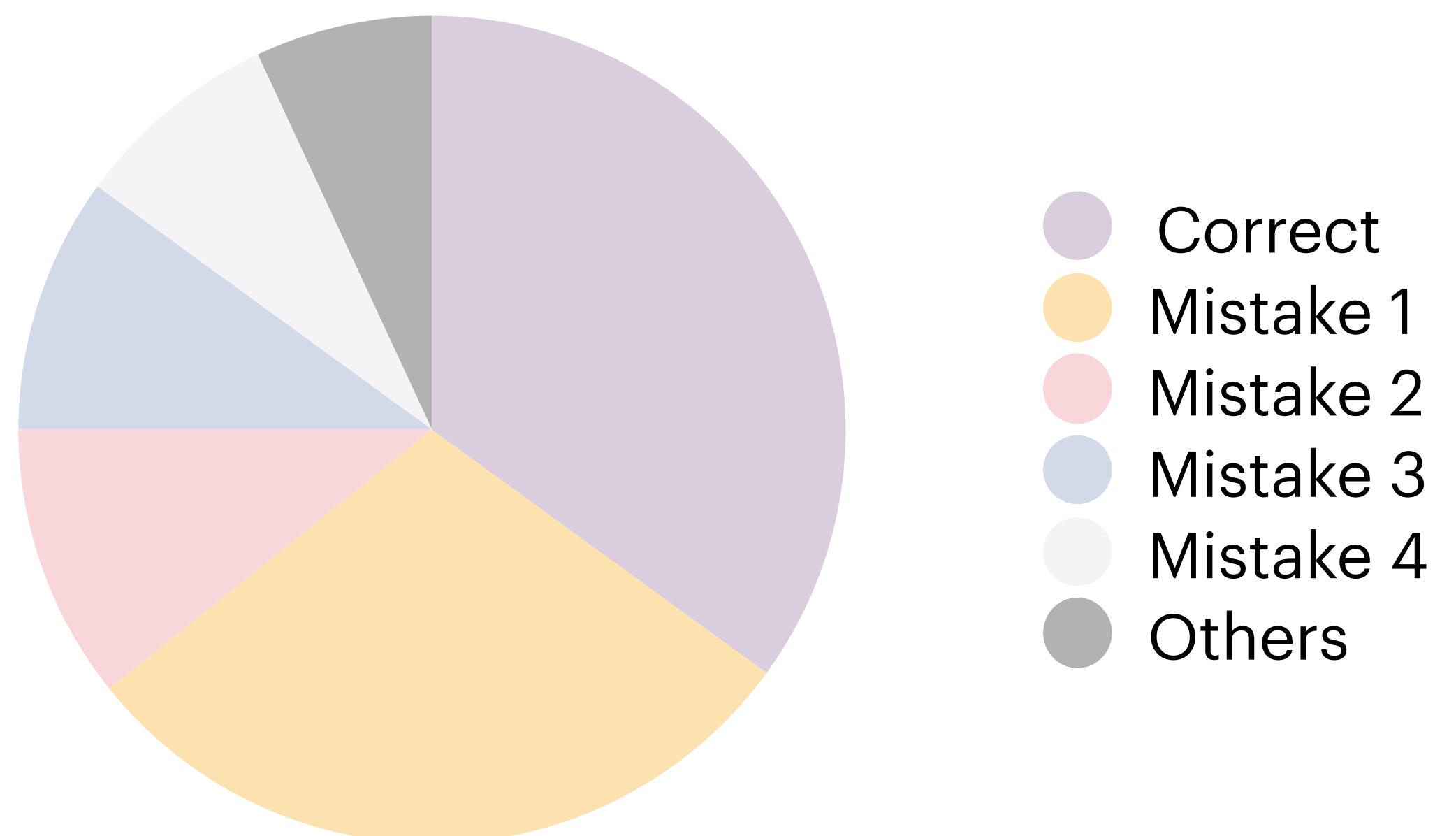
Instructors' Needs

Questions Instructors want to know

- How many students correctly write a loop?
- How many students use an anti-pattern that works but goes against the principles being taught?
- ...

Instructors' Needs

1. Find **common patterns and mistakes** in students' code
2. Get **descriptive statistics** about the class



Instructors' Needs

Questions Instructors want to know

- How many students correctly write a loop?
- How many students use an anti-pattern that works but goes against the principles being taught?
- ...

Instructors' Needs

1. Find **common patterns and mistakes** in students' code
2. Get **descriptive statistics** about the class
3. Find **code samples that include similar code lines** to an existing pattern

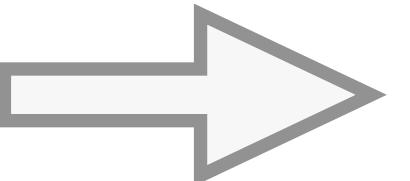
past_tense.append(word+'d')



Our Approach

Instructors' Needs

1. Find **common patterns and mistakes** in students' code
2. Get **descriptive statistics** about the class
3. Find **code samples that include similar code lines** to an existing pattern

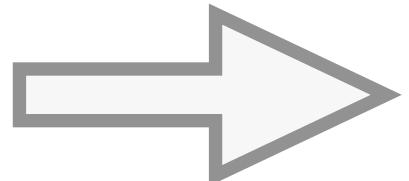


Code Search 

Our Approach

Instructors' Needs

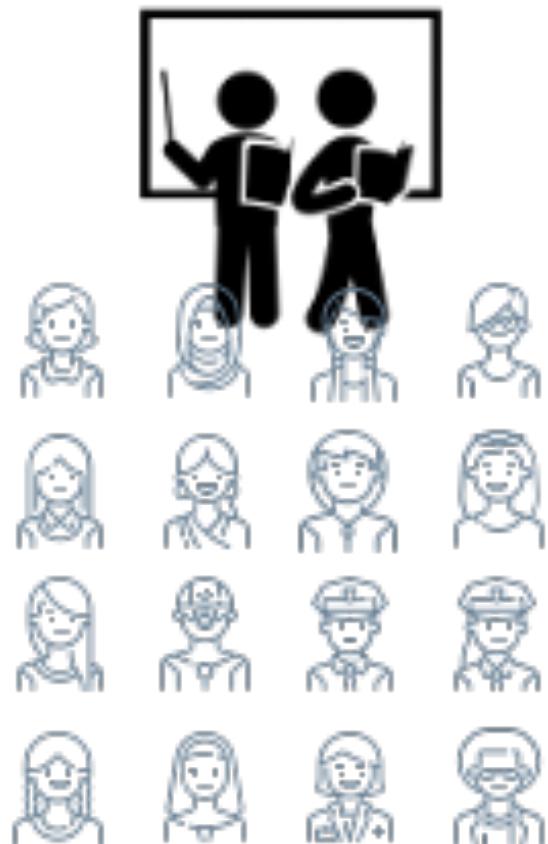
1. Find **common patterns and mistakes** in students' code
2. Get **descriptive statistics** about the class
3. Find **code samples that include similar code lines** to an existing pattern



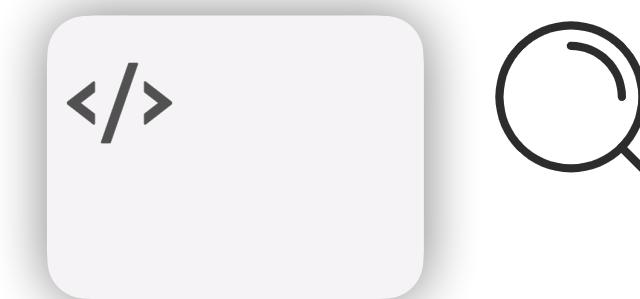
Code Search



Introductory programming education



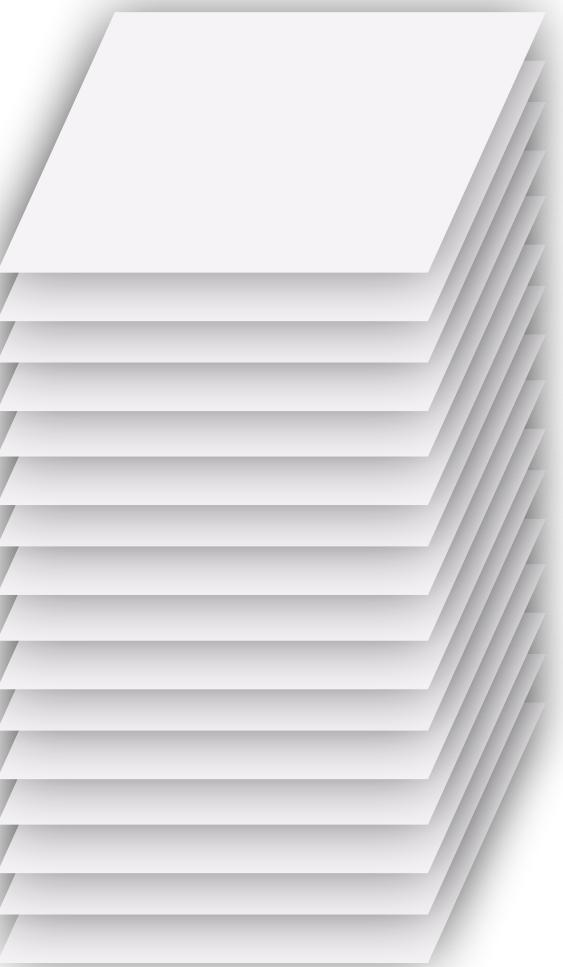
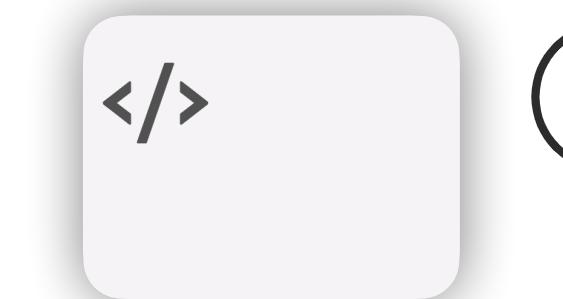
Code Search



Code search is a well-studied topic^[1]

[1] Di Grazia, & Pradel, M. Code search: A survey of techniques for finding code. ACM Comput. Surv 2023

Code Search



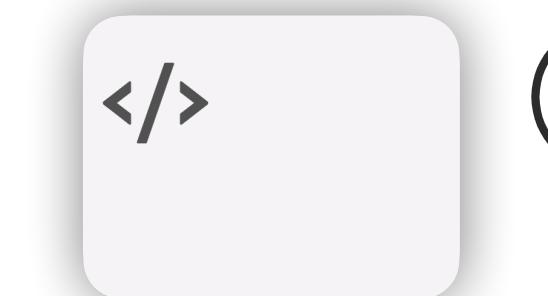
Code search is a well-studied topic^[1]



Locate specific parts of code
based on some search criteria

[1] Di Grazia, & Pradel, M. Code search: A survey of techniques for finding code. ACM Comput. Surv 2023

Code Search



Code search is a well-studied topic^[1]



Locate specific parts of code
based on some search criteria



Most of prior work focus on
professional developers [2, 3]

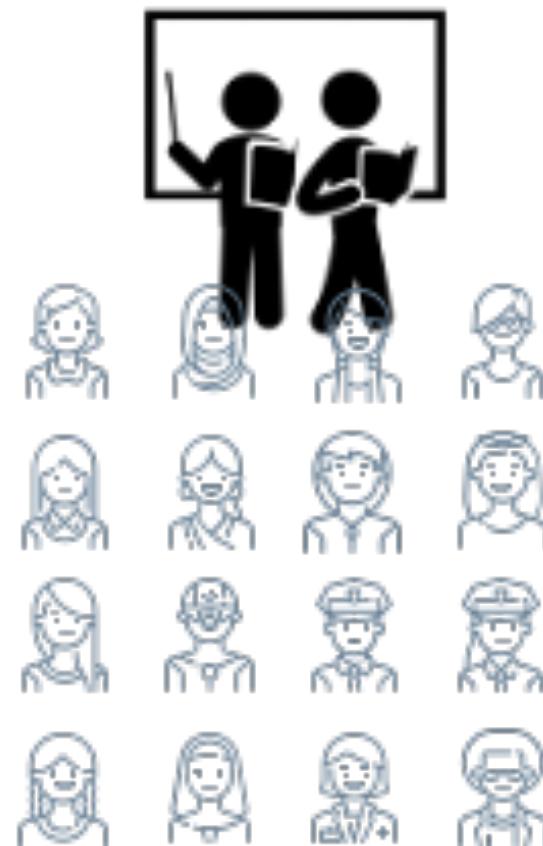
[1] Di Grazia, & Pradel, M. Code search: A survey of techniques for finding code. ACM Comput. Surv 2023

[2] Holmes et al., Using structural context to recommend source code examples. ICSE 2005

[3] Brandt et al., Example-centric programming: integrating web search into the development environment. CHI 2010

Code Search

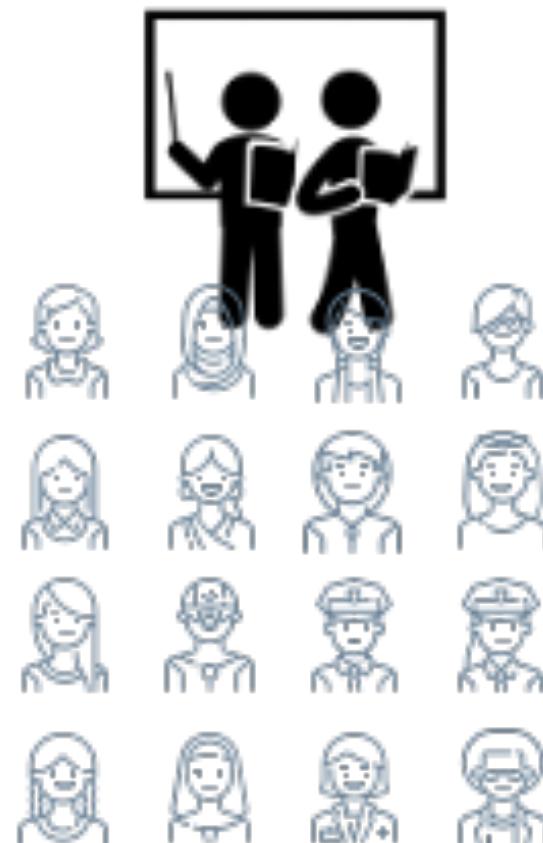
Introductory programming education



Unique challenges and opportunities for code search tools

Code Search

Introductory programming education

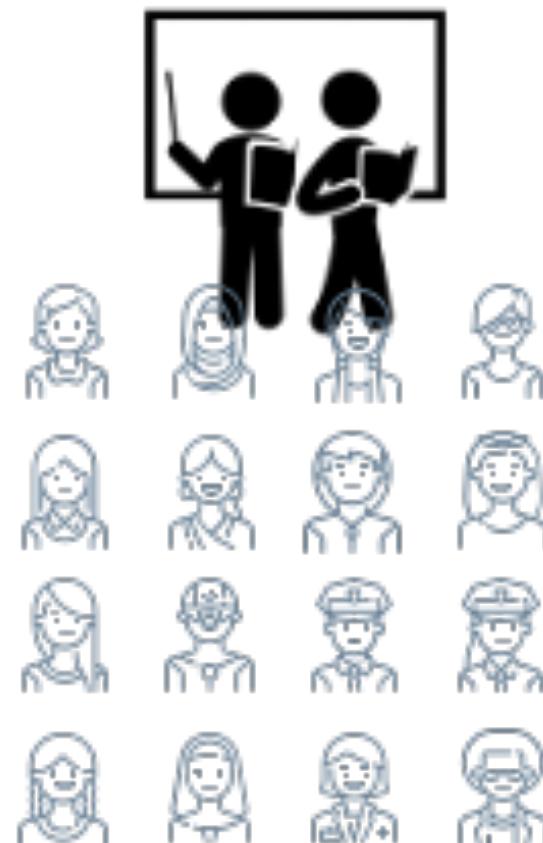


Unique challenges and opportunities for code search tools

Prior work is not sufficient in supporting programming instructors

Code Search

Introductory programming education



Unique challenges and opportunities for code search tools

Prior work is not sufficient in supporting programming instructors

Express meaningful and useful search criteria

Code Search in Introductory Programming Education

Adding a word to a list `past_tense`

```
past_tense = [] # a list of words  
...  
past_tense = past_tense + [new_word]
```

For each word in `words`, add 'd' to the end of the word if the word ends in 'e' to make it past tense. Otherwise, add 'ed' to make it past tense. Save these past tense words to a list called `past_tense`.

Code Search in Introductory Programming Education

Adding a word to a list `past_tense`

```
past_tense = [] # a list of words  
...  
past_tense = past_tense + [new_word]
```



A list that contains the new word. e.g.
["baked"], ["adapted"], ['grilled'] ...

For each word in `words`, add 'd' to the end of the word if the word ends in 'e' to make it past tense. Otherwise, add 'ed' to make it past tense. Save these past tense words to a list called `past_tense`.

Code Search in Introductory Programming Education

Adding a word to a list `past_tense`

```
past_tense = [] # a list of words  
...  
past_tense = past_tense + [new_word]
```



A list that contains the new word. e.g.
["baked"], ["adapted"], ['grilled'] ...

For each word in `words`, add 'd' to the end of the word if the word ends in 'e' to make it past tense. Otherwise, add 'ed' to make it past tense. Save these past tense words to a list called `past_tense`.

Text matching alone is not sufficient

Code Search in Introductory Programming Education

Adding a word to a list past_tense

```
past_tense = [] # a list of words  
...  
past_tense = past_tense + [new_word]
```

Multiple correct ways

A list that contains the new word. e.g.
["baked"], ["adapted"], ['grilled'] ...

Text matching alone is not sufficient

Code Search in Introductory Programming Education

Adding a word to a list past_tense

```
past_tense = [] # a list of words  
...  
past_tense = past_tense + [new_word]
```

Multiple correct ways

```
past_tense = past_tense + [word+'d']
```

A list that contains the new word. e.g.
["baked"], ["adapted"], ['grilled'] ...

Text matching alone is not sufficient

Code Search in Introductory Programming Education

Adding a word to a list past_tense

```
past_tense = [] # a list of words  
...  
past_tense = past_tense + [new_word]
```



A list that contains the new word. e.g.
["baked"], ["adapted"], ['grilled'] ...

Multiple correct ways

```
past_tense = past_tense + [word+'d']  
  
new_word = [word + 'd']  
...  
past_tense = past_tense + new_word
```

Text matching alone is not sufficient

Code Search in Introductory Programming Education

Adding a word to a list past_tense

```
past_tense = [] # a list of words  
...  
past_tense = past_tense + [new_word]
```



A list that contains the new word. e.g.
["baked"], ["adapted"], ['grilled'] ...

Text matching alone is not sufficient

Incorrect ways look similar to correct ones

```
past_tense = past_tense + [word+'d']
```

```
new_word = [word + 'd']
```

```
...  
past_tense = past_tense + new_word
```

```
new_word = word + 'd'
```

```
past_tense = past_tense + new_word
```

Code Search in Introductory Programming Education

Adding a word to a list `past_tense`

```
past_tense = [] # a list of words  
...  
past_tense = past_tense + [new_word]
```



A list that contains the new word. e.g.
["baked"], ["adapted"], ['grilled'] ...

Use Text Matching Alone is Insufficient

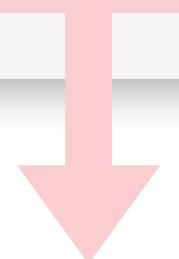
- Double quotation marks ('') and single quotation marks ("")

Text matching alone is not sufficient

Code Search in Introductory Programming Education

Adding a word to a list `past_tense`

```
past_tense = [] # a list of words  
...  
past_tense = past_tense + [new_word]
```



A list that contains the new word. e.g.
["baked"], ["adapted"], ['grilled'] ...

Use Text Matching Alone is Insufficient

- Double quotation marks ('') and single quotation marks ("")
- Variables can be defined elsewhere using various methods

Text matching alone is not sufficient

Code Search in Introductory Programming Education

Adding a word to a list `past_tense`

```
past_tense = [] # a list of words  
...  
past_tense = past_tense + [new_word]
```



A list that contains the new word. e.g.
["baked"], ["adapted"], ['grilled'] ...

Use Text Matching Alone is Insufficient

- Double quotation marks ('') and single quotation marks ("")
- Variables can be defined elsewhere using various methods
- Even if the variable names are in the same format, they could have **different values**

Text matching alone is not sufficient

Code Search in Introductory Programming Education

Adding a word to a list `past_tense`

```
past_tense = [] # a list of words  
...  
past_tense = past_tense + [new_word]
```



A list that contains the new word. e.g.
["baked"], ["adapted"], ['grilled'] ...

Use Text Matching Alone is Insufficient

- Double quotation marks ('') and single quotation marks ("")
- Variables can be defined elsewhere using various methods
- Even if the variable names are in the same format, they could have different values

Text matching: cumbersome and tedious

Code Search in Introductory Programming Education

Adding a word to a list past_tense

```
past_tense = [] # a list of words  
...  
past_tense = past_tense + [new_word]
```



A list that contains the new word. e.g.
["baked"], ["adapted"], ['grilled'] ...

Use Text Matching Alone is Insufficient

- Double quotation marks ('') and single quotation marks ("")
- Variables can be defined elsewhere using various methods
- Even if the variable names are in the same format, they could have different values

Text matching: cumbersome and tedious

Vector embeddings^[4]: unclear how reliable the results are

Code Search in Introductory Programming Education

Adding a word to a list past_tense

```
past_tense = [] # a list of words  
...  
past_tense = past_tense + [new_word]
```

RunEx: Regular Expression + Runtime values

```
past_tense = past_tense + << >>
```

A list that contains the new word. e.g.
["baked"], ["adapted"], ['grilled'] ...



Code Search in Introductory Programming Education

Adding a word to a list past_tense

```
past_tense = [] # a list of words  
...  
past_tense = past_tense + [new_word]
```

A list that contains the new word. e.g.
["baked"], ["adapted"], ['grilled'] ...

RunEx: Regular Expression + Runtime values

```
past_tense = past_tense + << >>
```

type(v) is list

len(v) == 1

Problem Characteristics

```

1
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = []
for word in words:
    newWord = word
    if(word[-1] == "e"):
        newWord += "d"
    else:
        newWord += "ed"
    past_tense.append(newWord)

print(words)
print(past_tense)

```



```

1
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = []
for word in words:
    newWord = word
    if(word[-1] == "e"):
        newWord += "d"
    else:
        newWord += "ed"
    past_tense.append(newWord)

print(words)
print(past_tense)

```



```

1
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = []
for word in words:
    newWord = word
    if(word[-1] == "e"):
        newWord += "d"
    else:
        newWord += "ed"
    past_tense.append(newWord)

print(words)
print(past_tense)

```



```

1
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = list()
for a in words:
    if a[-1] == 'e':
        old_time = a + 'd'
    else:
        old_time = a + 'ed'
    past_tense.append(old_time)

print(past_tense)

```

Students' solutions are **short and self-contained**

Problem Characteristics

```
1
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = []
for word in words:
    newWord = word
    if(word[-1] == "e"):
        newWord += "d"
    else:
        newWord += "ed"
    past_tense.append(newWord)

print(words)
print(past_tense)
```

```
1
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = []
for word in words:
    newWord = word
    if(word[-1] == "e"):
        newWord += "d"
    else:
        newWord += "ed"
    past_tense.append(newWord)

print(words)
print(past_tense)
```

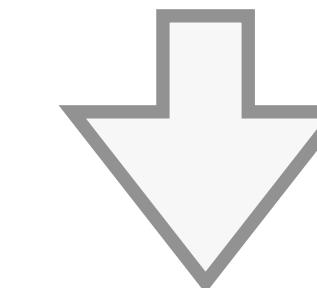
```
1
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = []
for word in words:
    newWord = word
    if(word[-1] == "e"):
        newWord += "d"
    else:
        newWord += "ed"
    past_tense.append(newWord)

print(words)
print(past_tense)
```

```
1
words = ["adopt", "bake", "beam", "confide", "grit"]
past_tense = list()
for a in words:
    if a[-1] == 'e':
        old_time = a + 'd'
    else:
        old_time = a + 'ed'
    past_tense.append(old_time)

print(past_tense)
```

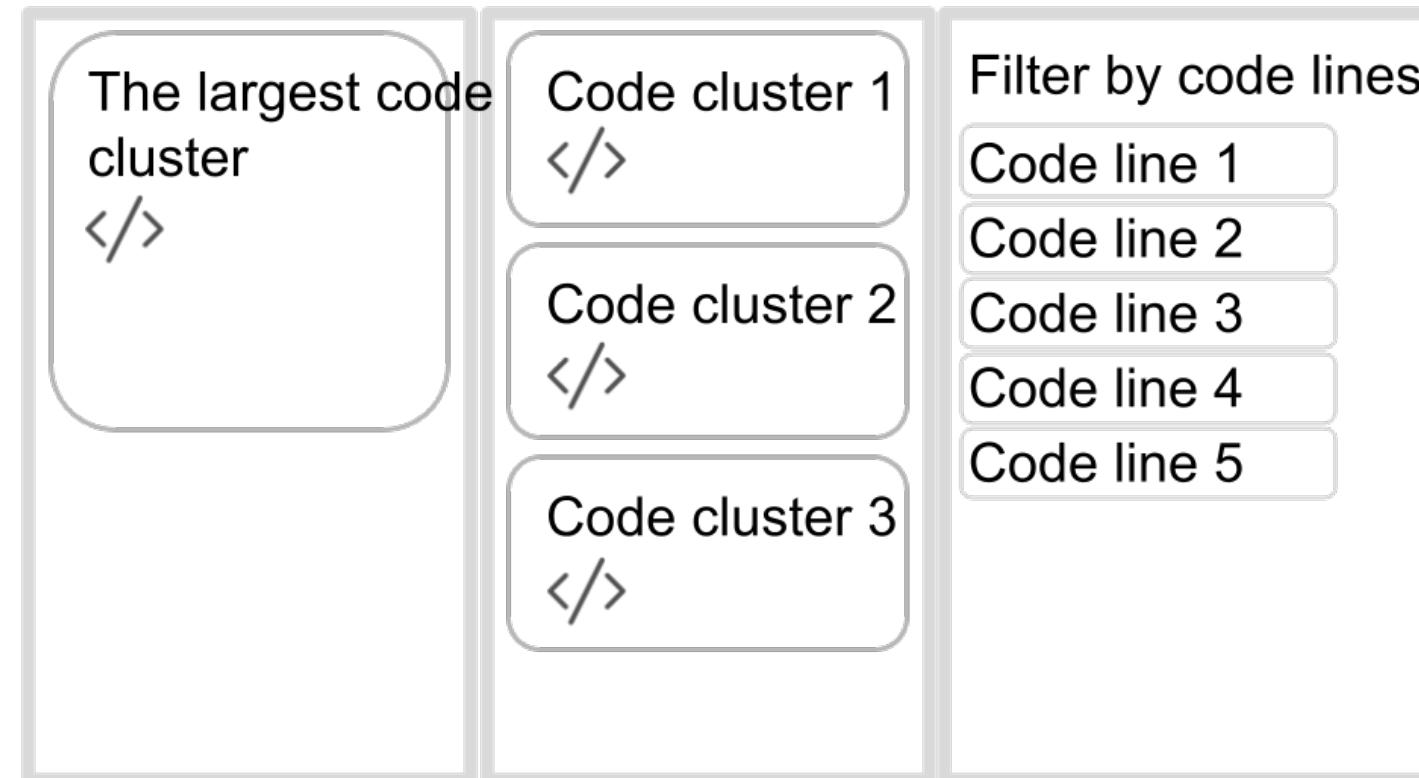
Students' solutions are **short and self-contained**



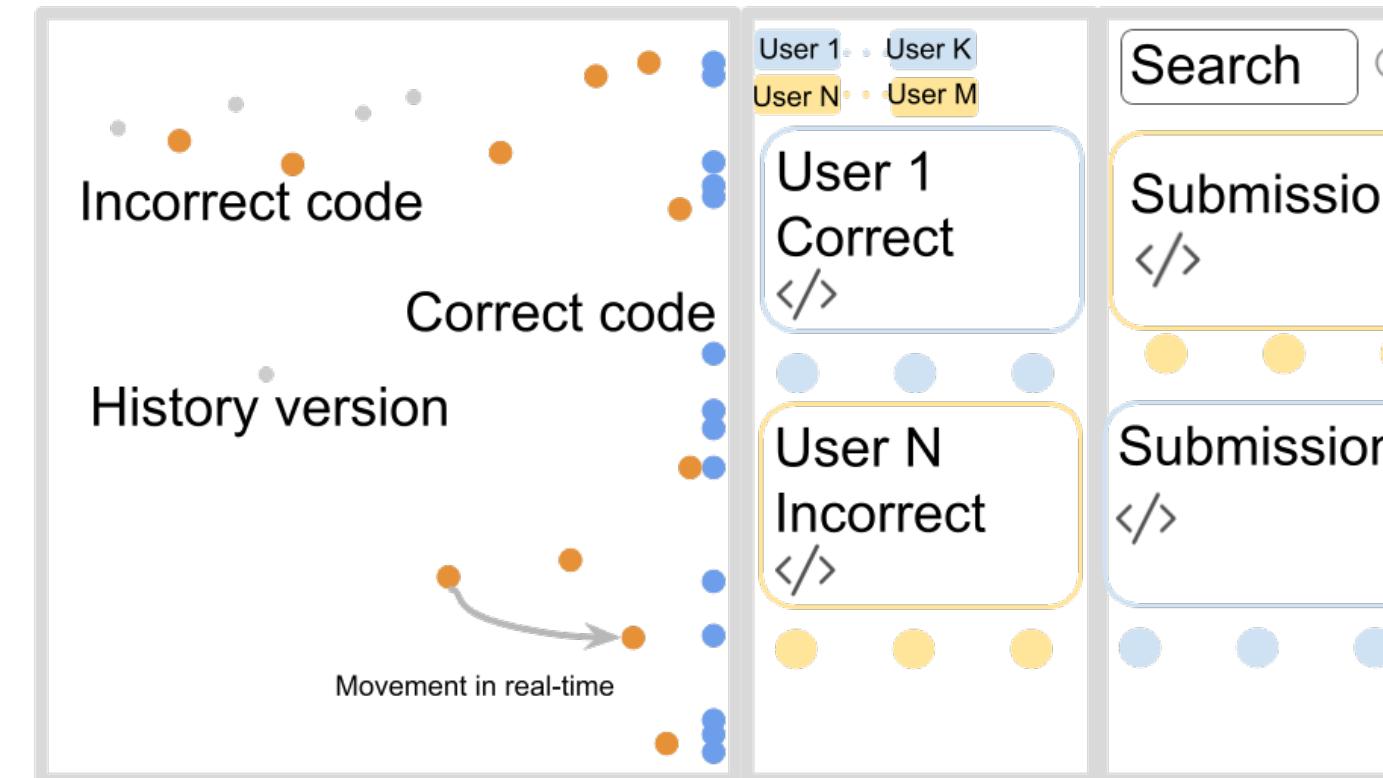
Execute the code samples to search across **runtime values**

Programming Education at Scale

OverCode^[5]



VizProg^[6]

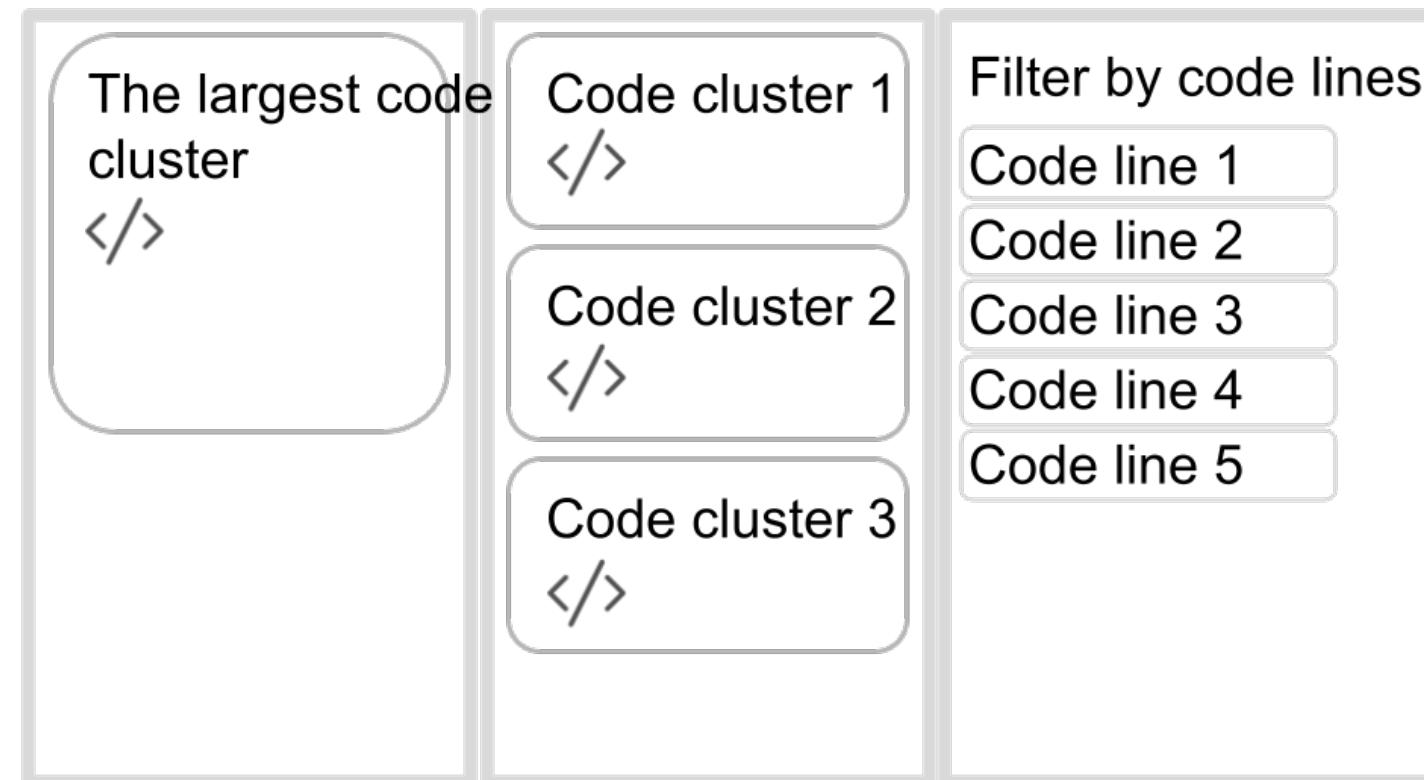


[5] Glassman et al., OverCode: Visualizing variation in student solutions to programming problems at scale. TOCHI 2015

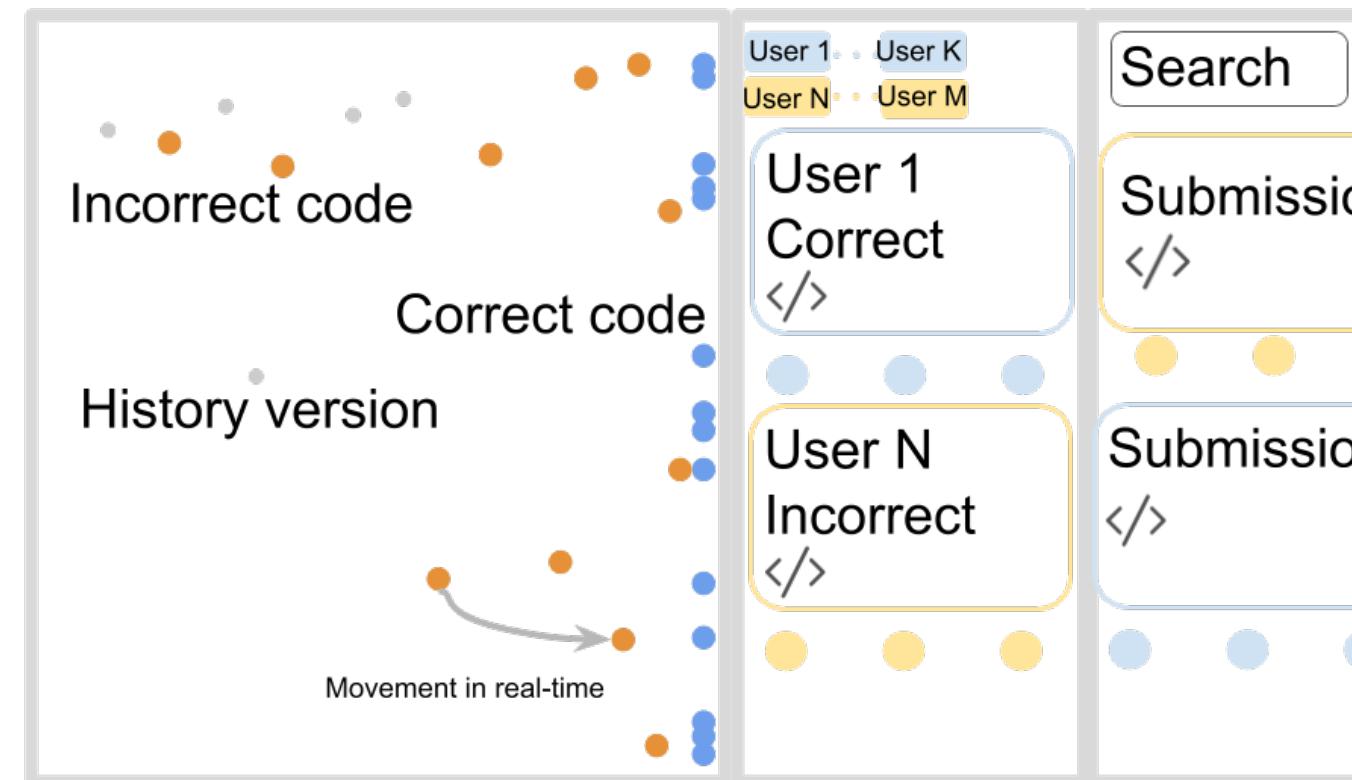
[6] Zhang et al., VizProg: Identifying Misunderstandings By Visualizing Students' Coding Progress. CHI 2023

Programming Education at Scale

OverCode^[5]



VizProg^[6]



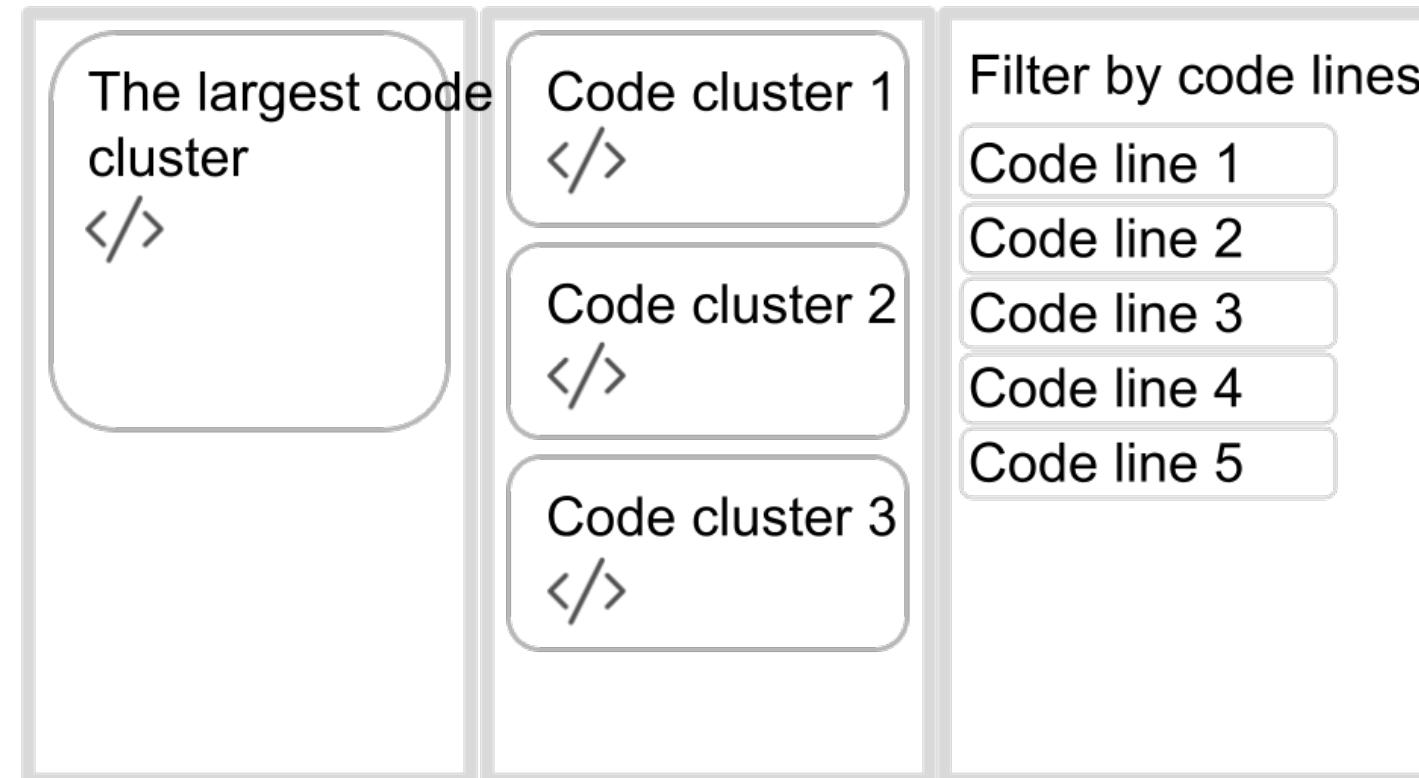
Presents **clusters** of students' code by grouping solutions that have exactly **same computation process**

[5] Glassman et al., OverCode: Visualizing variation in student solutions to programming problems at scale. TOCHI 2015

[6] Zhang et al., VizProg: Identifying Misunderstandings By Visualizing Students' Coding Progress. CHI 2023

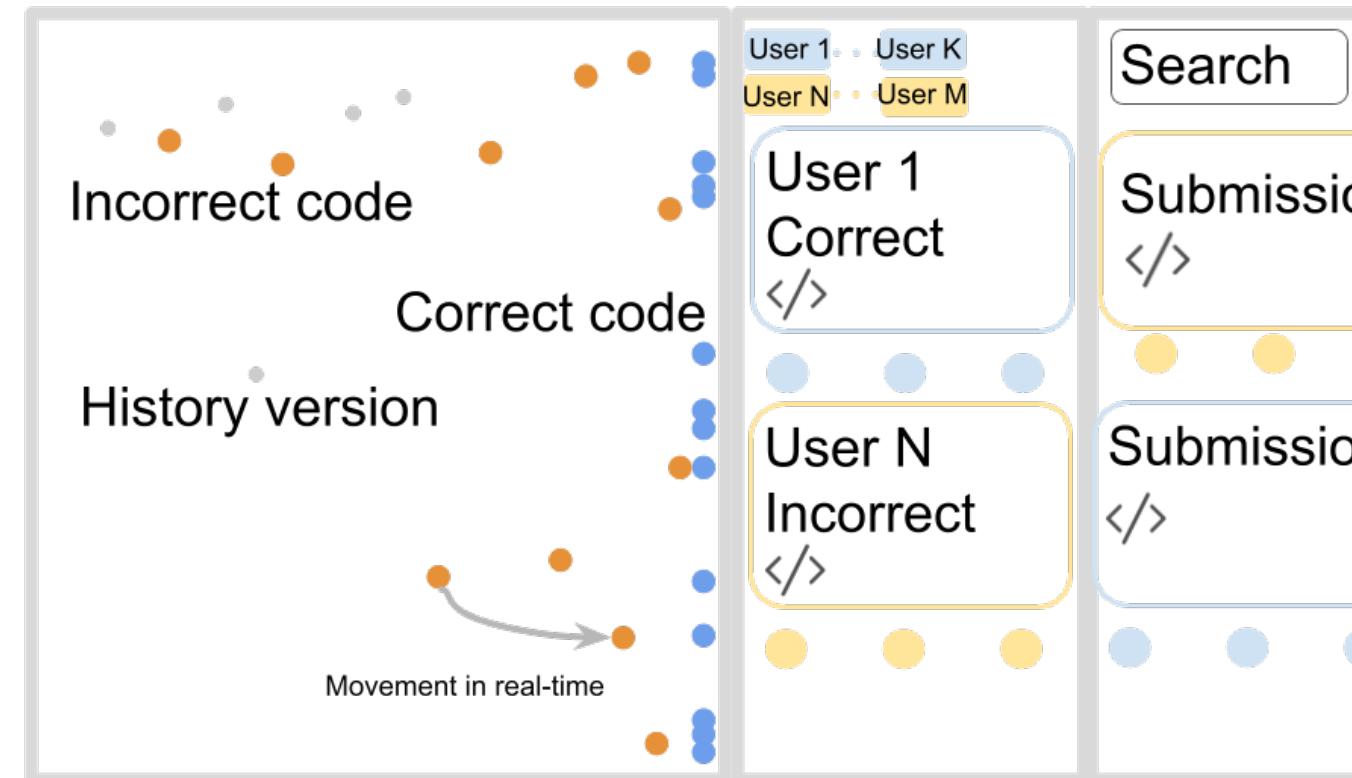
Programming Education at Scale

OverCode^[5]



Presents **clusters** of students' code by grouping solutions that have exactly **same computation process**

VizProg^[6]

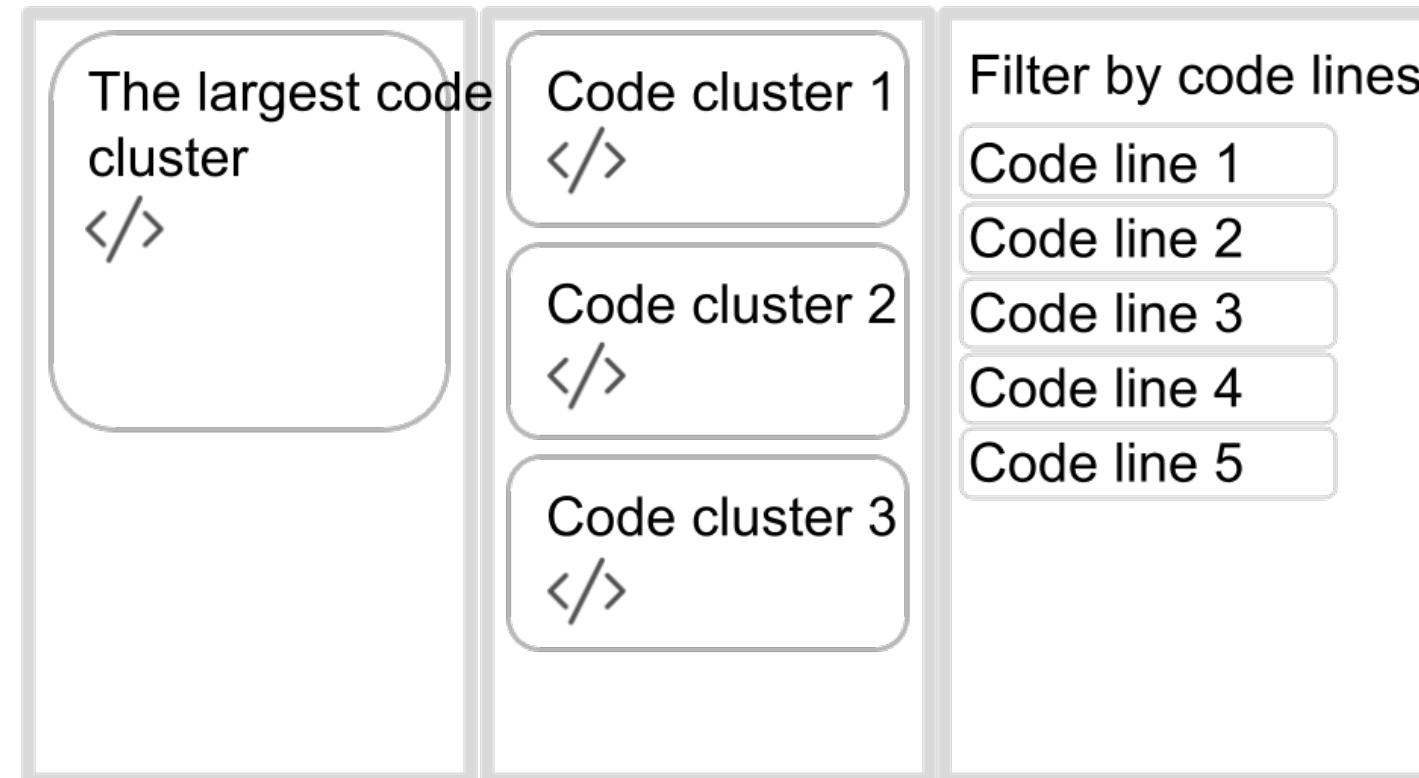


Uses **dynamic dots** to present students' coding progress on **a 2D map**

- [5] Glassman et al., OverCode: Visualizing variation in student solutions to programming problems at scale. TOCHI 2015
[6] Zhang et al., VizProg: Identifying Misunderstandings By Visualizing Students' Coding Progress. CHI 2023

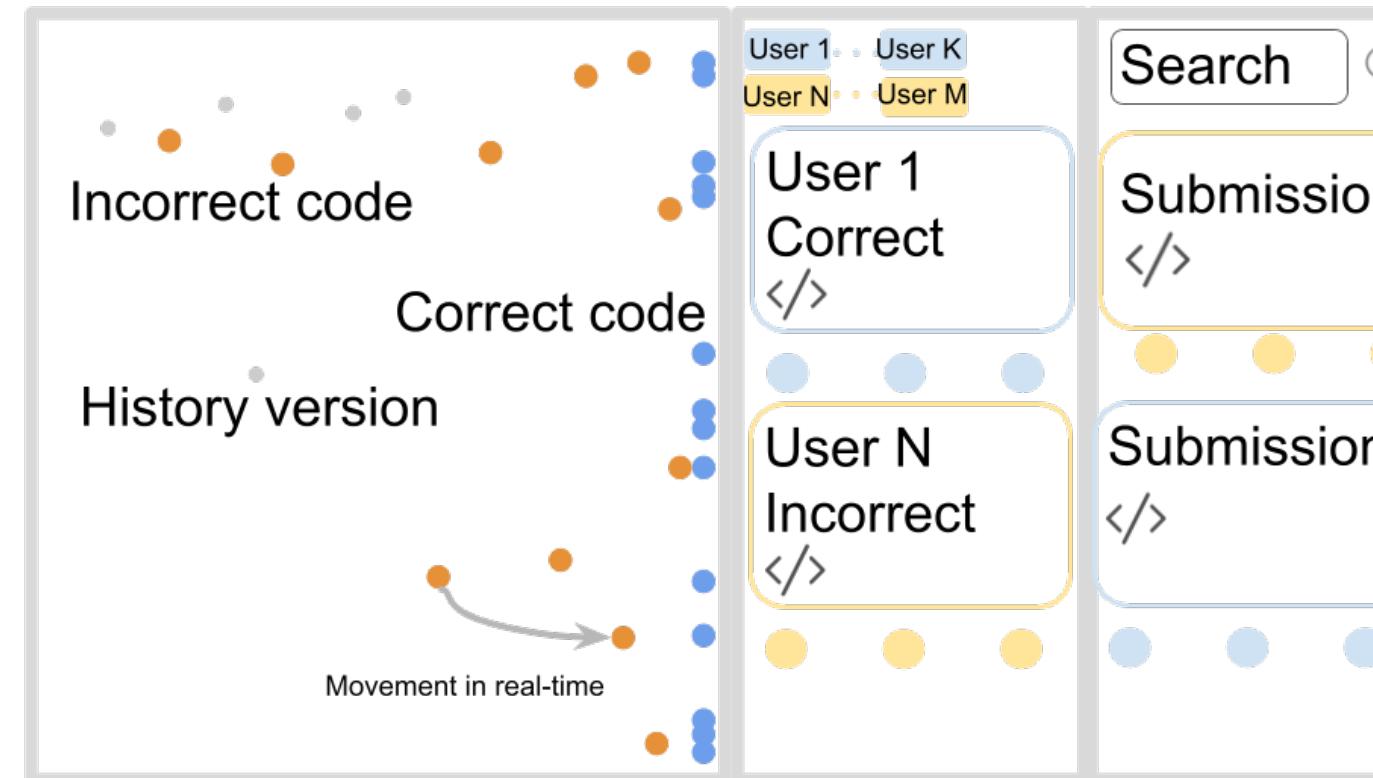
Programming Education at Scale

OverCode^[5]



Presents **clusters** of students' code by grouping solutions that have exactly **same computation process**

VizProg^[6]



Uses **dynamic dots** to present students' coding progress on **a 2D map**



Instructors have **little control** over the results

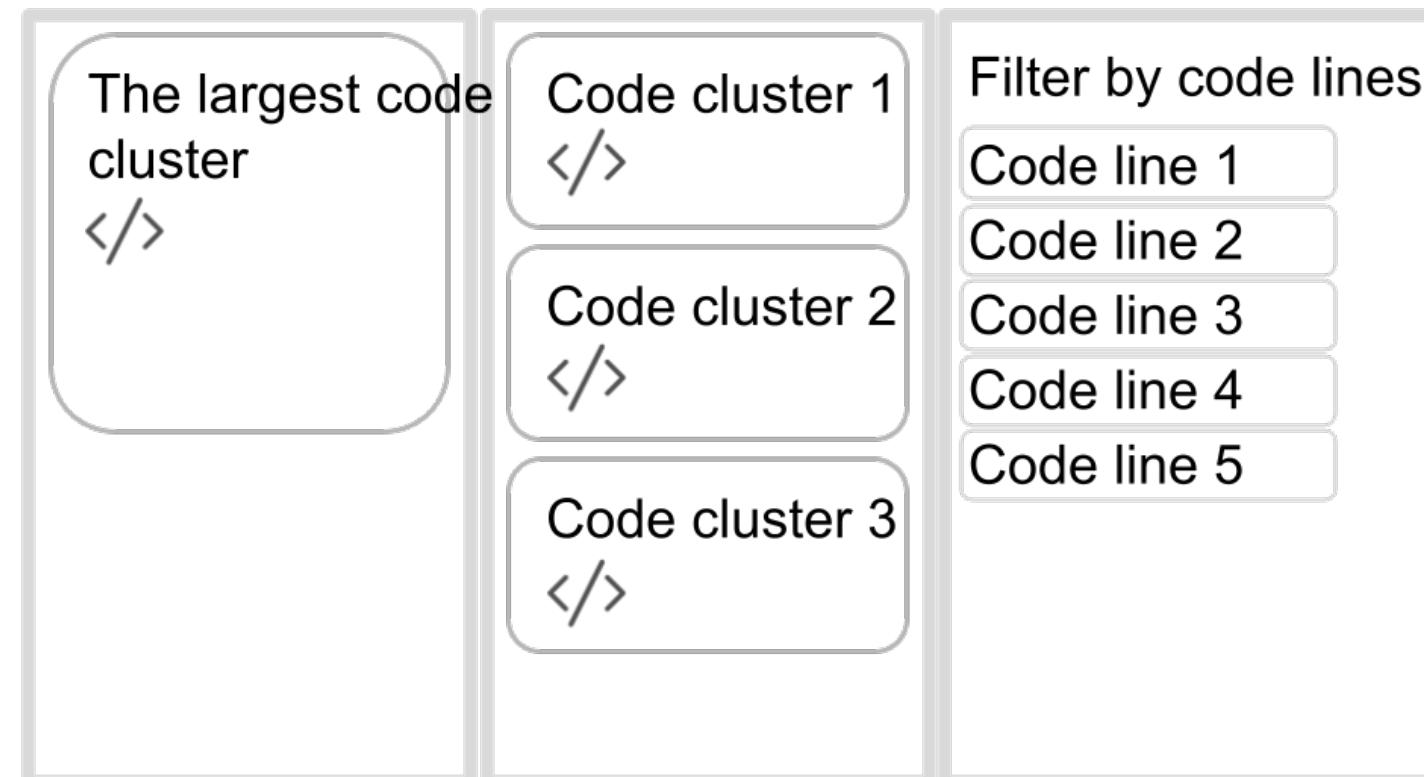
What is presented may not be what instructors want to know



- [5] Glassman et al., OverCode: Visualizing variation in student solutions to programming problems at scale. TOCHI 2015
[6] Zhang et al., VizProg: Identifying Misunderstandings By Visualizing Students' Coding Progress. CHI 2023

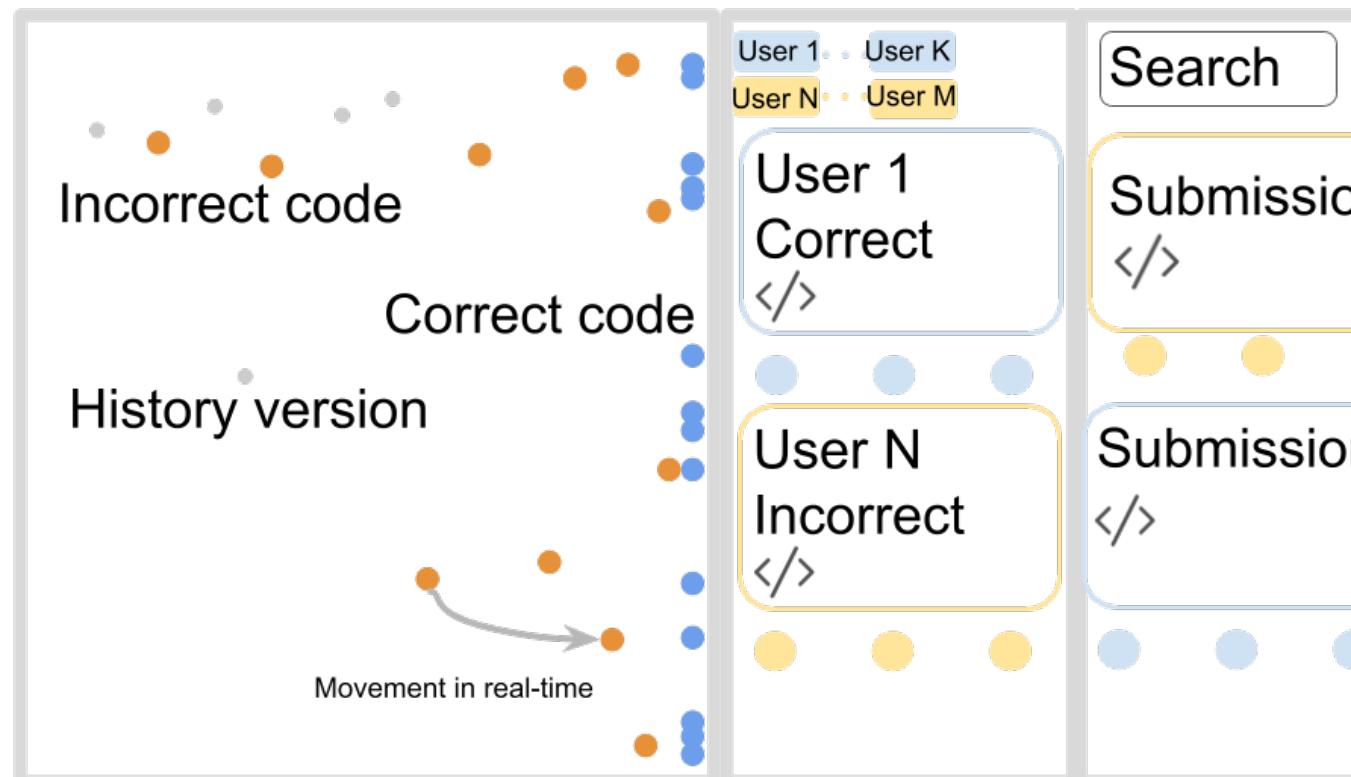
Programming Education at Scale

OverCode^[5]



Presents **clusters** of students' code by grouping solutions that have exactly **same computation process**

VizProg^[6]



Uses **dynamic dots** to present students' coding progress on **a 2D map**

RunEx(this paper)

Allow instructors to **customize the summarization** of students' code by creating their own search queries to filter solutions

- [5] Glassman et al., OverCode: Visualizing variation in student solutions to programming problems at scale. TOCHI 2015
[6] Zhang et al., VizProg: Identifying Misunderstandings By Visualizing Students' Coding Progress. CHI 2023

Code Search for Programming Education at Scale

Codewebs^[7]

Abstract Syntax Trees
(ASTs)

+

Unit Test Results

RunEx(this paper)

[7] Nguyen et al., Codewebs: scalable homework search for massive open online programming courses. WWW 2014

Code Search for Programming Education at Scale

Codewebs^[7]

Abstract Syntax Trees
(ASTs)

+

Unit Test Results

- **Hold after** code samples have executed

RunEx(this paper)

[7] Nguyen et al., Codewebs: scalable homework search for massive open online programming courses. WWW 2014

Code Search for Programming Education at Scale

Codewebs^[7]

Abstract Syntax Trees
(ASTs)

+

Unit Test Results

- **Hold after** code samples have executed

RunEx(this paper)

- Filter by **intermediate values**

[7] Nguyen et al., Codewebs: scalable homework search for massive open online programming courses. WWW 2014

Code Search for Programming Education at Scale

Codewebs^[7]

Abstract Syntax Trees
(ASTs)

+

Unit Test Results

- **Hold after** code samples have executed

RunEx(this paper)

- Filter by **intermediate values**
- Check how a student solve problem more accurately than Codewebs

[7] Nguyen et al., Codewebs: scalable homework search for massive open online programming courses. WWW 2014



Syntax

How does RunEx augment regular expression
with runtime values?

1

Syntax

How does RunEx augment regular expression with runtime values?

2

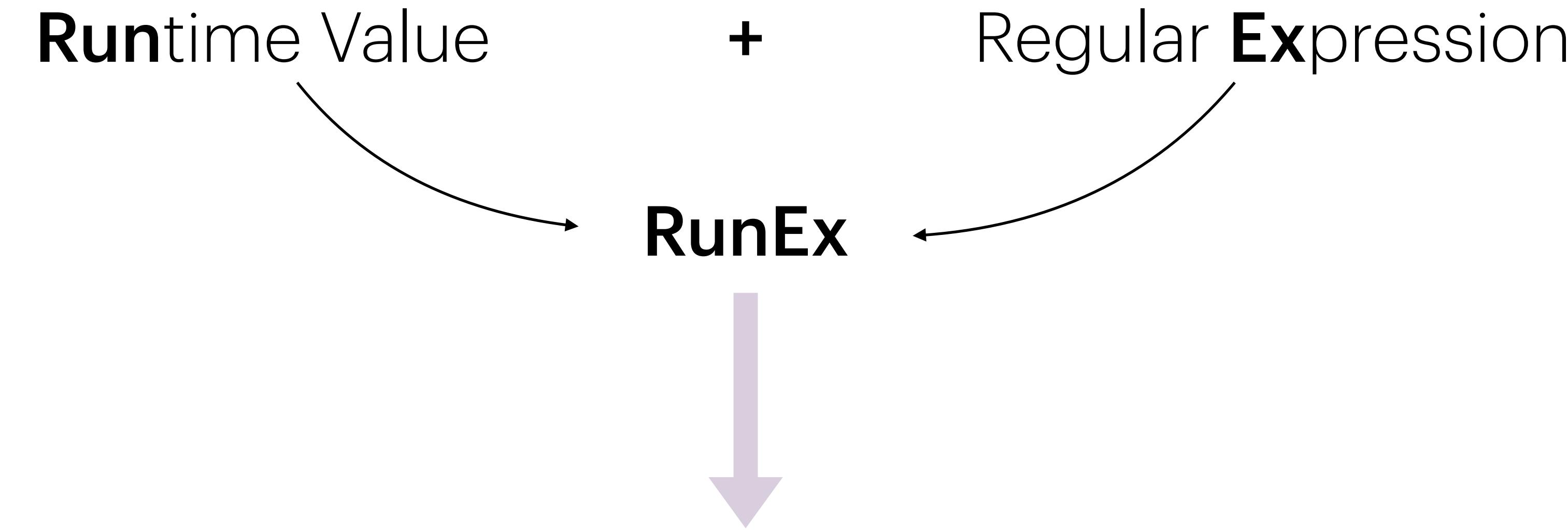
User Interface

How do users create search queries in RunEx?

Syntax



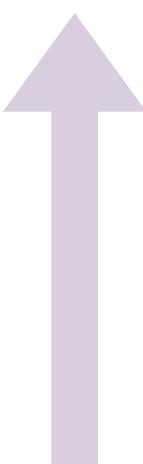
Syntax



Greater expressiveness and precision
Maintain the flexibility of Regex

Syntax

Runtime Value

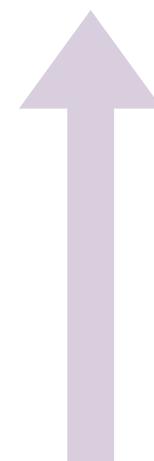


Constraints

<<...>>

Syntax

Runtime Value



Constraints

<<...>>

Create a list ranging from 0 to 39 using `range`

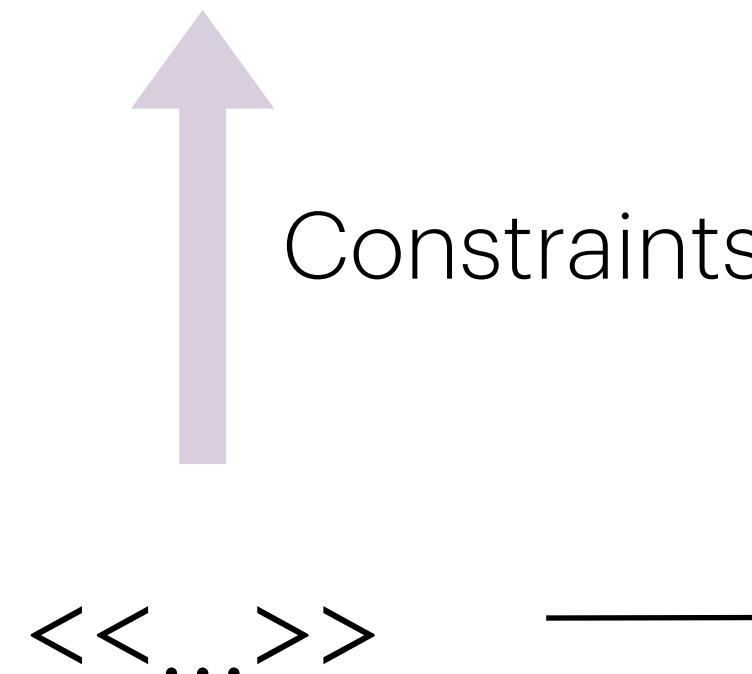
`range(param1)`



Runtime value of param1
should be 40

Syntax

Runtime Value



Create a list ranging from 0 to 39 using `range`

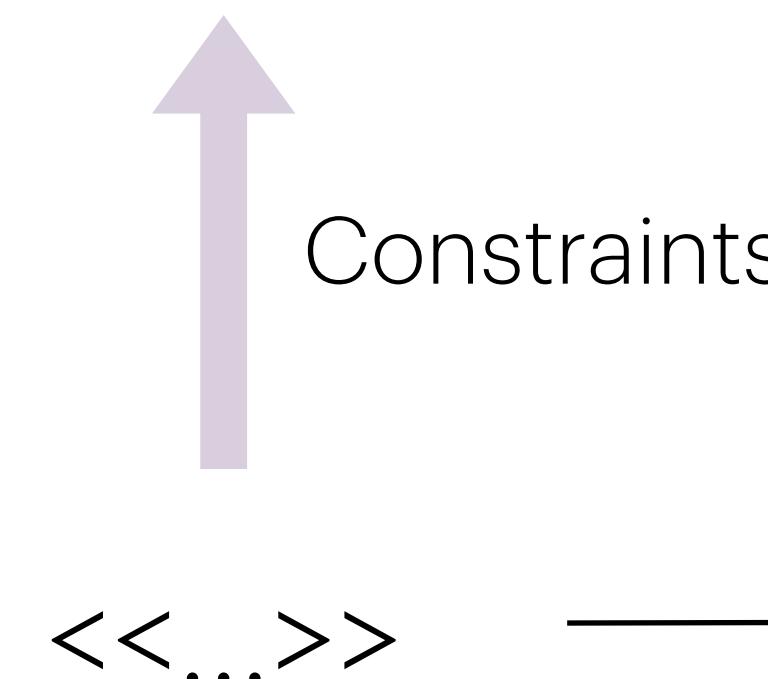
`range(param1)`



Runtime value of param1
should be 40

Syntax

Runtime Value



Regular **E**xpression

Create a list ranging from 0 to 39 using **range**

range(param1)



Runtime value of param1
should be 40

range(<<40>>)

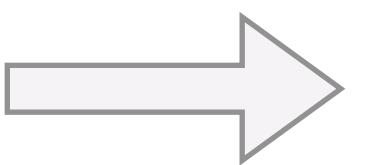


range(any variable or expression)

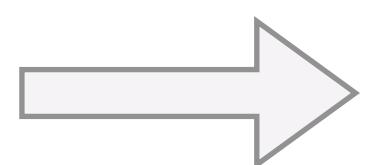
Syntax

Runtime Value

+



range(<<40>>)



Regular **E**xpression

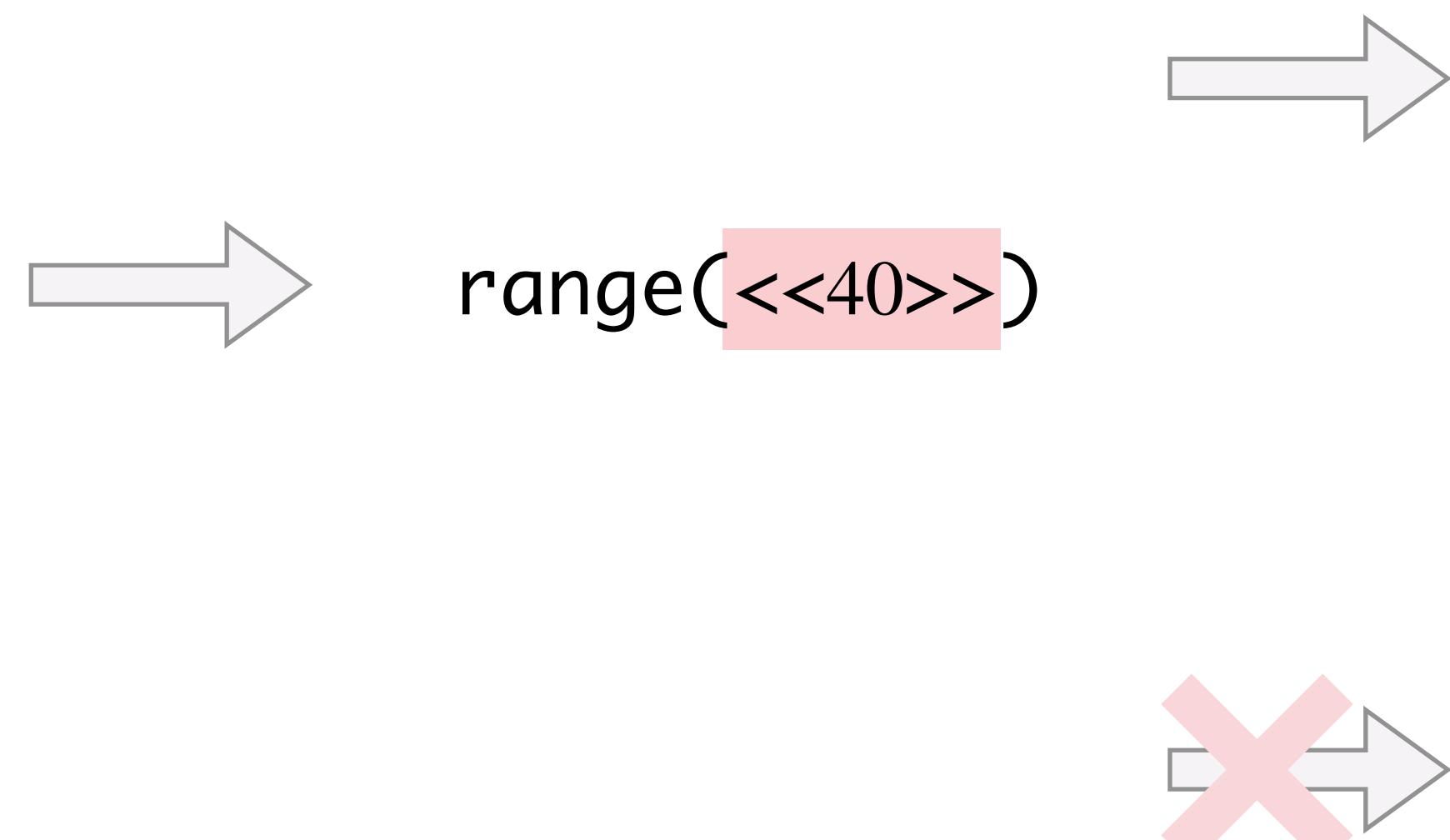
range(20+20)

K = 40
range(K)

range(40)

Syntax

Runtime Value
+
Regular Expression



range(20+20)

K = 40
range(K)

range(40)

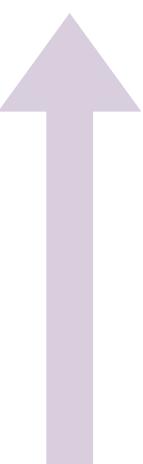
range(39)

last_item = 39
range(last_item)

range(40-1)

Syntax

Runtime Value



Constraints

<<...>>

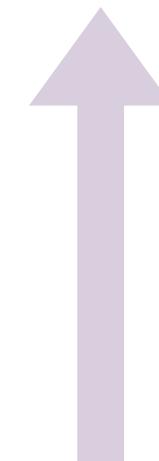
Syntax

Runtime Value



Syntax

Runtime Value



Constraints

<<...>>

Python conditions

Append a word to an existing list

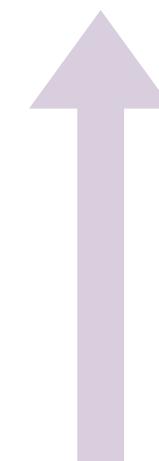
```
past_tense = past_tense + [new_word]
```



A list that only has one word
in it

Syntax

Runtime Value



Constraints

<<...>> ————— Python conditions —————>

Append a word to an existing list

```
past_tense = past_tense + [new_word]
```

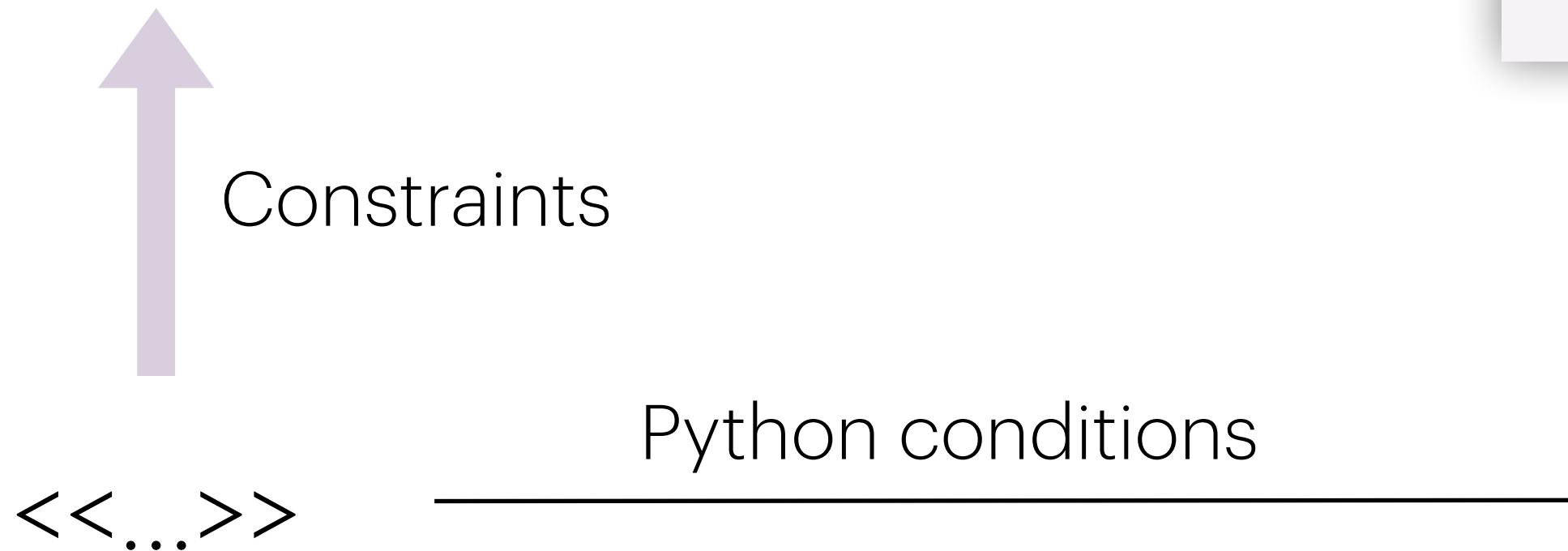


A list that only has one word
in it

```
past_tense = past_tense + <<λv>>
```

Syntax

Runtime Value



Append a word to an existing list

```
past_tense = past_tense + [new_word]
```

A list that only has one word
in it

```
past_tense = past_tense + <<λv>>
```

```
lambda v: type(v) is list and len(v)==1
```

Implementation

```
SIGNAL = False
def __EVAL__(variable, fn):
    global SIGNAL
    if fn(variable):
        SIGNAL = True
    return variable
```

```
sum = 0
for i in range(<<40>>):
    sum += i
```

Inject into the original code

Implementation

```
SIGNAL = False
def __EVAL__(variable, fn):
    global SIGNAL
    if fn(variable):
        SIGNAL = True
    return variable
```

```
sum = 0
for i in range(<<40>>):
    sum += i
```

Inject into the original code

```
sum = 0
for i in range(__EVAL__(old_v, lambda v: v==40)):
    sum += i
```

User Interface

Correct Incorrect

if i in counts :

if v0 not in counts :

counts . get (v0 , 0)

 s1 = "hello"
counts={}
for i in s1:
 counts[i]=0
for i in s1:
 counts[i]+=1 s1 = "hello"
counts = {}
for s in s1:
 counts[s] += s1.count(s)

KeyError: h on line 4

 s1 = "hello"
counts = {}
for s in s1:
 counts[s] = s1.count(s) s1 = "hello"
counts = {}
for ch in s1:
 if ch not in counts: count
 counts[ch] += 1 s1 = "hello"
counts = {}
for c in s1:
 counts[c] = counts.get(c, s1 = "hello"
counts = {}
for letter in s1:
 if letter in counts:
 counts[letter] += 1
 else:

User Interface

The screenshot shows a user interface for running and testing code. At the top, there are two buttons: "Correct" with a checked checkbox and "Incorrect" with a checked checkbox. Below these buttons is a search bar with a magnifying glass icon.

The main area displays several code snippets, each with a magnifying glass icon to its right. The snippets are:

- `s1 = "hello"`
- `counts = {}`
- `for s in s1:`
- `counts[s] = s.count(s)`

Below this snippet is a red error message box containing the text "KeyError: h on line 4".

Other snippets shown include:

- `s1 = "hello"`
- `counts = {}`
- `for ch in s1:`
- `if ch not in counts: count`
- `counts[ch] += 1`

At the bottom, another snippet is shown:

- `s1 = "hello"`
- `counts = {}`
- `for c in s1:`
- `counts[c] = counts.get(c,`

Search Query View

User Interface

```
if i in counts :  
if v0 not in counts :  
counts . get ( v0 , 0 )
```

Correct Incorrect

Code List View

```
s1 = "hello"  
counts={}  
for i in s1:  
    counts[i]=0  
for i in s1:  
    counts[i]+=1
```

```
s1 = "hello"  
counts = {}  
for s in s1:  
    counts[s] += s1.count(s)
```

KeyError: h on line 4

```
s1 = "hello"  
counts = {}  
for s in s1:  
    counts[s] = s1.count(s)
```

```
s1 = "hello"  
counts = {}  
for ch in s1:  
    if ch not in counts: count  
    counts[ch] += 1
```

```
s1 = "hello"  
counts = {}  
for c in s1:  
    counts[c] = counts.get(c,
```

```
s1 = "hello"  
counts = {}  
for letter in s1:  
    if letter in counts:  
        counts[letter] += 1  
    else:
```

User Interface

The screenshot shows the RunEx user interface with several code editor panes and a search bar.

Top Left Editor:

```
if i in counts:  
    if v0 not in counts:  
        counts . get ( v0 , 0 )
```

Top Right Editor (Correct):

```
s1 = "hello"  
counts={}  
for i in s1:  
    counts[i]=0  
for i in s1:  
    counts[i]+=1
```

Top Right Editor (Incorrect):

```
s1 = "hello"  
counts = {}  
for s in s1:  
    counts[s] += s1.count(s)
```

A red box highlights the error message: **KeyError: h on line 4**.

Middle Left Editor:

```
s1 = "hello"  
counts = {}  
for s in s1:  
    counts[s] = s1.count(s)
```

Middle Right Editor:

```
s1 = "hello"  
counts = {}  
for ch in s1:  
    if ch not in counts: count  
    counts[ch] += 1
```

Bottom Left Editor:

```
s1 = "hello"  
counts = {}  
for c in s1:  
    counts[c] = counts.get(c,
```

Bottom Right Editor:

```
s1 = "hello"  
counts = {}  
for letter in s1:  
    if letter in counts:  
        counts[letter] += 1  
    else:
```

Search Bar:

Correct Incorrect

User Interface

The screenshot shows the RunEx user interface with several code editor panes and a results panel.

Results Panel:

- Correct Incorrect
- KeyError: h on line 4

Code Editor Panes:

- Top Left: if i in counts :
if v0 not in counts :
counts . get (v0 , 0)
- Top Right: s1 = "hello"
counts={}
for i in s1:
counts[i]=0
for i in s1:
counts[i]+=1
- Middle Left: s1 = "hello"
counts = {}
for s in s1:
counts[s] = s1.count(s)
- Middle Right: s1 = "hello"
counts = {}
for ch in s1:
if ch not in counts: count
counts[ch] += 1
- Bottom Left: s1 = "hello"
counts = {}
for c in s1:
counts[c] = counts.get(c,
- Bottom Right: s1 = "hello"
counts = {}
for letter in s1:
if letter in counts:
counts[letter] += 1
else:

User Interface

The screenshot shows the RunEx user interface with several code editor panes and a sidebar.

Top Bar: Contains a search icon and a refresh icon.

Toolbar: Shows "Correct" and "Incorrect" status indicators with checked checkboxes.

Panels:

- Left Panel:** Displays three snippets of Python code:
 - `if i in counts:`
 - `if v0 not in counts:` (highlighted in red)
 - `counts . get (v0 , 0)`
- Center Panel:** Shows a comparison between two snippets:
 - Left snippet:`s1 = "hello"
counts={}
for i in s1:
 counts[i]=0
for i in s1:
 counts[i]+=1`
 - Right snippet:`s1 = "hello"
counts = {}
for s in s1:
 counts[s] += s1.count(s)`A red box highlights the error message "KeyError: h on line 4".
- Bottom Panels:** Show two more snippets:
 - `s1 = "hello"
counts = {}
for s in s1:
 counts[s] = s1.count(s)` (highlighted in orange)
 - `s1 = "hello"
counts = {}
for ch in s1:
 if ch not in counts: count
 counts[ch] += 1`

User Interface

The screenshot displays the RunEx user interface with several windows and panels:

- Top Left Panel:** Shows a code editor with three lines of Python code:

```
if i in counts:  
if v0 not in counts:  
counts . get ( v0 , 0 )
```
- Top Right Panel:** A status bar with "Correct" and "Incorrect" buttons, each with a checked checkbox.
- Middle Left Panel:** A code editor window showing:

```
s1 = "hello"  
counts={}  
for i in s1:  
    counts[i]=0  
for i in s1:  
    counts[i]+=1
```
- Middle Right Panel:** A code editor window showing:

```
s1 = "hello"  
counts = {}  
for s in s1:  
    counts[s] += s1.count(s)
```

A red box highlights the error message "KeyError: h on line 4".
- Bottom Left Panel:** A code editor window showing:

```
s1 = "hello"  
counts = {}  
for s in s1:  
    counts[s] = s1.count(s)
```

A dropdown menu is open over the last line, with options: "value (v):", "name (n):", and "match any variable name" with a checked checkbox.
- Bottom Right Panel:** A code editor window showing:

```
s1 = "hello"  
counts = {}  
for letter in s1:  
    if letter in counts:  
        counts[letter] += 1  
    else:
```

User Interface

The screenshot displays the RunEx user interface with several components:

- Code Editor:** On the left, there is a code editor window containing Python code. The code includes a loop that initializes a dictionary `counts` and updates it based on a string `s1`. A tooltip for the variable `v0` is shown, stating: "value (v): type(v) is str && len(v)==1" and "name (n):".
- Execution Results:** In the center, there are two execution results. The top one shows a `KeyError: h` on line 4, indicating an error in the user's code. The bottom one shows the correct output of the program.
- Toolbars and Buttons:** At the top and bottom of the interface are toolbars with various icons for file operations, search, and help.

User Interface

The screenshot shows the RunEx user interface with several code editor panes and a search bar.

Top Left Editor:

```
if i in counts:  
if v0 not in counts:  
counts . get ( v0 , 0 )
```

Top Right Editor (Correct):

```
s1 = "hello"  
counts={}  
for i in s1:  
    counts[i]=0  
for i in s1:  
    counts[i]+=1
```

Top Right Editor (Incorrect):

```
s1 = "hello"  
counts = {}  
for s in s1:  
    counts[s] += s1.count(s)
```

Bottom Left Editor:

```
s1 = "hello"  
counts = {}  
for char in s1:  
    value (v):  
    name (n): len(n) > 1  
match any variable name
```

Bottom Right Editor:

```
s1 = "hello"  
counts = {}  
for ch in s1:  
    if ch not in counts: count  
    counts[ch] += 1
```

Search Bar:

```
s1 = nello  
counts = {}  
for c in s1:  
    counts[c] = counts.get(c,
```

Buttons:

- Correct
- Incorrect

User Interface

The screenshot displays the RunEx user interface with several components:

- Code Editor:** On the left, there is a code editor window containing Python code. The code includes a loop that initializes a dictionary `counts` and updates it based on a string `s1`. A tooltip is visible over the line `counts[s] = s1.count(s)`, showing its type as `value (v): type(v) is str && len(v)==1`.
- Execution Results:** In the center, there are two execution results. The top one shows a `KeyError: h` on line 4, indicating an error in the user's code. The bottom one shows the correct output of the program.
- Toolbars and Buttons:** At the top and bottom of the interface are toolbars with various icons for file operations, search, and help.

User Interface

Correct ✓ Incorrect ✓

The image shows a user interface for running and testing Python code. On the left is a code editor with several lines of code. The first line is a comment. The second line contains an if statement with a condition that is highlighted in red. The third line is another comment. The fourth line contains a call to the get method on the counts dictionary, with the argument v0 highlighted in blue. The fifth line is a comment. The sixth line contains an assignment statement where counts[v0] is assigned the value s1.count(v0).

Results Panel:

- Correct ✓**: Shows the code and its execution result.
- Incorrect ✓**: Shows the code and an error message: `KeyError: h on line 4`.
- Code Completion Panel (Bottom Left):** Shows the code with a cursor at the end of the counts[s] = s1.count(s) line. A tooltip provides information about the value(v) function: `value (v): type(v) is str && len(v)==1`. It also shows the name(n) and match any variable name checkboxes.
- Code Completion Panel (Bottom Right):** Shows the code with a cursor at the end of the counts[s] += 1 line. A tooltip provides information about the value(v) function: `value (v): type(v) is str && len(v)==1`. It also shows the name(n) and match any variable name checkboxes.

```

if i in counts:
    counts[i] += 1
else:
    counts[i] = 1
counts[v0] = s1.count(v0)
print(counts)

```

User Interface

Correct Incorrect

The screenshot shows a code editor on the left and a results panel on the right.

Code Editor:

```

if i in counts:
    if v0 not in counts:
        counts . get ( v0 , 0 )
counts [ v0 ] = s1 . count ( v0 )

```

Results Panel:

Shows two examples of code execution:

- Example 1:** A correct snippet of Python code that counts occurrences of characters in a string `s1` and stores the counts in a dictionary `counts`. It handles the case where a character is not present in the dictionary by returning 0. The result is displayed in a light gray box.
- Example 2:** An incorrect snippet of Python code that attempts to count characters in a string `s1` and store the counts in a dictionary `counts`. It uses a for loop over `s` instead of `ch` and tries to update the dictionary with `counts[s] += s1.count(s)`, which raises a `KeyError` for the character 'h'. The result is displayed in a dark red box with the error message "KeyError: h on line 4".

Below the results panel, there are two more code snippets with dropdown menus showing variable definitions:

```

s1 = "hello"
counts = {}
for s in s1:
    counts[s] = s1.count(s)

```

```

s1 = "hello"
counts = {}
for ch in s1:
    if ch not in counts: count
        value (v): type(v) is str && len(v)==1
        name (n):
        match any variable name 

```

At the bottom, there are two more code snippets:

```

s1 = "hello"
counts = {}
for c in s1:
    counts[c] = counts.get(c,

```

```

s1 = "hello"
counts = {}
for letter in s1:
    if letter in counts:
        counts[letter] += 1
    else:

```

User Interface

Correct Incorrect

The image shows a screenshot of the RunEx user interface. On the left is a dark grey code editor window with a light grey border. It contains the following Python code:if i in counts:
if v0 not in counts:
counts . get (v0 , 0)A red rounded rectangle highlights the line `counts . get (v0 , 0)`. In the top right corner of the editor are two small icons: a red 'X' and a magnifying glass.

Below the editor are six light grey preview cards, each containing a different version of the same code. Each card has a magnifying glass icon in its top right corner.

- Card 1:

```
s1 = "hello"  
counts = {}  
for s in s1:  
    counts[v0] = s1.count(v0)
```
- Card 2:

```
s1 = "hello"  
counts={}  
for i in s1:  
    counts[i]=s1.count(i)
```
- Card 3:

```
s1 = "hello"  
set_s1 = set(list(s1))  
counts = {}  
for s in set_s1:  
    counts[s] = s1.count(s)
```
- Card 4:

```
s1 = "hello"  
counts = {}  
for x in set(s1):  
    counts[x] = s1.count(x)
```
- Card 5:

```
s1 = "hello"  
counts = {}  
for ch in s1:  
    counts[ch] = s1.count(ch)
```
- Card 6:

```
s1 = "hello"  
counts = {}  
for letter in s1:  
    counts[letter] = s1.count(  
print(freq)
```

At the bottom of the page, there is a pink horizontal bar with the text "NameError: name 'freq' is not defined".

User Interface

The screenshot shows the RunEx user interface with a code editor on the left and execution results on the right.

Code Editor (Left):

```
if i in counts:  
if v0 not in counts:  
counts . get ( v0 , 0 )  
counts [ v0 ] = s1 . count ( v0 )
```

Execution Results (Right):

Correct Incorrect

Four execution cards are shown:

- Card 1:** Shows a partially completed code snippet:

```
s1 = "hello"  
counts={}  
for i in s1:  
    counts[i]=0  
for i in s1:  
    counts[i]+=1
```

A red box highlights the error: **KeyError: h on line 4**.
- Card 2:** Shows a correct code snippet:

```
s1 = "hello"  
counts = {}  
for s in s1:  
    counts[s] = s1.count(s)
```
- Card 3:** Shows another correct code snippet:

```
s1 = "hello"  
counts = {}  
for ch in s1:  
    if ch not in counts: count  
    counts[ch] += 1
```
- Card 4:** Shows a correct code snippet:

```
s1 = "hello"  
counts = {}  
for letter in s1:  
    if letter in counts:  
        counts[letter] += 1  
    else:
```

Set Maths on Queries

Correct Incorrect

`s1 = "hello"`
`counts = {}`
`for s in s1:`
 `counts[s] += 1`

`s1 = "hello"`
`counts = {}`
`for s in s1:`
 `counts[s] += s1.count(s)`

KeyError: h on line 4

`s1 = "hello"`
`counts = {}`
`for ch in s1:`
 `if ch not in counts: count`
 `counts[ch] += 1`

`s1 = "hello"`
`counts = {}`
`for c in s1:`
 `counts[c] = counts.get(c,`

86

Set Maths on Queries

Correct Incorrect

`s1 = "hello"`
`counts = {}`
`for s in s1:`
 `counts[s] += 1`

`s1 = "hello"`
`counts = {}`
`for s in s1:`
 `counts[s] += s1.count(s)`

KeyError: h on line 4

`s1 = "hello"`
`counts = {}`
`for ch in s1:`
 `if ch not in counts: count`
 `counts[ch] += 1`

`s1 = "hello"`
`counts = {}`
`for c in s1:`
 `counts[c] = counts.get(c,`

87

Set Maths on Queries

Correct Incorrect

```
s1 = "hello"
counts = {}
for i in s1:
    counts[i]=0
for i in s1:
    counts[i]+=1
```

KeyError: h on line 4

```
s1 = "hello"
counts = {}
for s in s1:
    counts[s] += s1.count(s)
```

```
s1 = "hello"
counts = {}
for ch in s1:
    if ch not in counts: count
    counts[ch] += 1
```

```
s1 = "hello"
counts = {}
for c in s1:
    counts[c] = counts.get(c,
                           0) + 1
```

```
s1 = "hello"
counts = {}
for letter in s1:
    if letter in counts:
        counts[letter] += 1
    else:
```

Set Maths on Queries

Correct Incorrect

<code>counts [v0] = s1 . count (v0)</code>	<code>s1 = "hello" counts={} for i in s1: counts[i]=0 for i in s1: counts[i]+=1</code>	<code>s1 = "hello" counts = {} for s in s1: counts[s] += s1.count(s)</code>
<code>if i in counts :</code>	<code>KeyError: h on line 4</code>	<code>s1 = "hello" counts = {} for ch in s1: if ch not in counts: count counts[ch] += 1</code>
<code>if v0 not in counts :</code>		
<code>counts . get (v0 , 0)</code>		
<code>counts [v0] = s1 . count (v0)</code>	<code>s1 = "hello" counts = {} for s in s1: counts[s] = s1.count(s)</code>	<code>s1 = "hello" counts = {} for letter in s1: if letter in counts: counts[letter] += 1 else:</code>

Set Maths on Queries

Correct Incorrect

```
s1 = "hello"
counts = {}
for i in s1:
    counts[i]=0
for i in s1:
    counts[i]+=1
```

KeyError: h on line 4

```
s1 = "hello"
counts = {}
for s in s1:
    counts[s] += s1.count(s)
```

```
s1 = "hello"
counts = {}
for ch in s1:
    if ch not in counts: count
    counts[ch] += 1
```

```
s1 = "hello"
counts = {}
for c in s1:
    counts[c] = counts.get(c,
                           0) + 1
```

```
s1 = "hello"
counts = {}
for letter in s1:
    if letter in counts:
        counts[letter] += 1
    else:
```

Set Maths on Queries

Correct Incorrect

The image shows a programming exercise interface with several code snippets and their execution results.

- Top Left:** A code editor window with a blue border. Inside, there's a red 'X' icon and a magnifying glass icon. The code is:


```
counts [ v0 ] = s1 . count ( v0 )
```
- Middle Left:** A list of code snippets with circular icons:
 - `if i in counts :`
 - `if v0 not in counts :` (A hand cursor icon is over this line)
 - `counts . get (v0 , 0)`
 - `counts [v0] = s1 . count (v0)`
- Background Snippets:** Several code snippets are displayed in the background, each with a magnifying glass icon:
 - `s1 = "hello"`
`counts={}`
`for i in s1:`
 `counts[i]=0`
`for i in s1:`
 `counts[i]+=1`
 - `s1 = "hello"`
`counts = {}`
`for s in s1:`
 `counts[s] += s1.count(s)`
 - `s1 = "hello"`
`counts = {}`
`for ch in s1:`
 `if ch not in counts: count`
 `counts[ch] += 1`
 - `s1 = "hello"`
`counts = {}`
`for letter in s1:`
 `if letter in counts:`
 `counts[letter] += 1`
 `else:`
- Bottom Right:** A red box highlights an error message: `KeyError: h on line 4`.

Set Maths on Queries

Correct Incorrect

```

s1 = "hello"
counts = {}
for i in s1:
    counts[i]=0
for i in s1:
    counts[i]+=1

```

KeyError: h on line 4

```

s1 = "hello"
counts = {}
for s in s1:
    counts[s] += s1.count(s)

```

```

s1 = "hello"
counts = {}
for ch in s1:
    if ch not in counts: count
    counts[ch] += 1

```

```

s1 = "hello"
counts = {}
for letter in s1:
    if letter in counts:
        counts[letter] += 1
    else:

```

Set Maths on Queries

Correct Incorrect

The screenshot shows a programming exercise interface with several code snippets:

- Top Left:** A snippet with a red 'X' icon and a magnifying glass icon. It contains the line: `counts [v0] = s1 . count (v0)`.
- Second Column Top:** A snippet starting with `if v0 not in counts :`
- Third Column Top:** A snippet starting with `s1 = "hello"` followed by a loop to initialize a counts dictionary.
- Second Column Middle:** A snippet starting with `if i in counts :`
- Third Column Middle:** A snippet starting with `for s in s1:` followed by a loop to update the counts dictionary.
- Second Column Bottom:** A snippet starting with `if v0 not in counts :`
- Third Column Bottom:** A snippet starting with `counts . get (v0 , 0)` followed by a loop to update the counts dictionary.
- Bottom Left:** A snippet starting with `counts [v0] = s1 . count (v0)`.
- Bottom Middle:** A snippet starting with `s1 = "hello"` followed by a loop to initialize a counts dictionary.
- Bottom Right:** A snippet starting with `for ch in s1:` followed by a loop to update the counts dictionary.

A red box highlights the error message `KeyError: h on line 4` in the third column middle section, indicating that the key 'h' was not found in the empty dictionary.

Set Maths on Queries

X Q
Correct Incorrect

(if i in counts :

(if v0 not in counts :

(counts . get (v0 , 0)

(counts [v0] = s1 . count (v0)

counts [v0] = s1 . count (v0)

|| if v0 not in counts :

s1 = "hello"
 counts = {}

 for c in s1:
 if c not in counts:
 counts[c] = 0
 counts[c] += 1

s1 = "hello"
 counts = {}
 for c in s1:
 if c not in counts:
 counts[c] = 1
 else:
 counts[c] +=1

s1 = "hello"
 counts = {}
 for s in s1:
 counts[s] = s1.count(s)

s1 = "hello"
 counts = {}
 for ch in s1:
 if ch not in counts: count
 counts[ch] += 1

s1 = "hello"
 counts = {}
 for a in s1:
 if a not in counts:

s1 = "hello"
 counts={}
 for i in s1:
 counts[i]=s1.count(i)

Chain Queries

Correct Incorrect

Code Snippet	Test Case	Result
<pre>s1 = "hello" counts = {} for i in s1: counts[i]=0 for i in s1: counts[i]+=1</pre>	<pre>counts [v0] = 0</pre>	KeyError: h on line 4
<pre>if v0 not in counts: counts [v0] = 1</pre>	<pre>if v0 not in counts:</pre>	KeyError: h on line 4
<pre>if v0 not in counts: counts [v0] = 1</pre>	<pre>counts [v0] = 1</pre>	KeyError: h on line 4
<pre>if v0 not in counts: counts [v0] = 0</pre>	<pre>counts [v0] = 0</pre>	KeyError: h on line 4

Chain Queries

Conditional statement

counts [v0] = 0

if v0 not in counts :

counts [v0] = 1

if v0 not in counts :

counts [v0] = 1

if v0 not in counts :

counts [v0] = 0

Correct Incorrect

s1 = "hello"
counts={}
for i in s1:
 counts[i]=0
for i in s1:
 counts[i]+=1

s1 = "hello"
counts = {}
for s in s1:
 counts[s] += s1.count(s)

KeyError: h on line 4

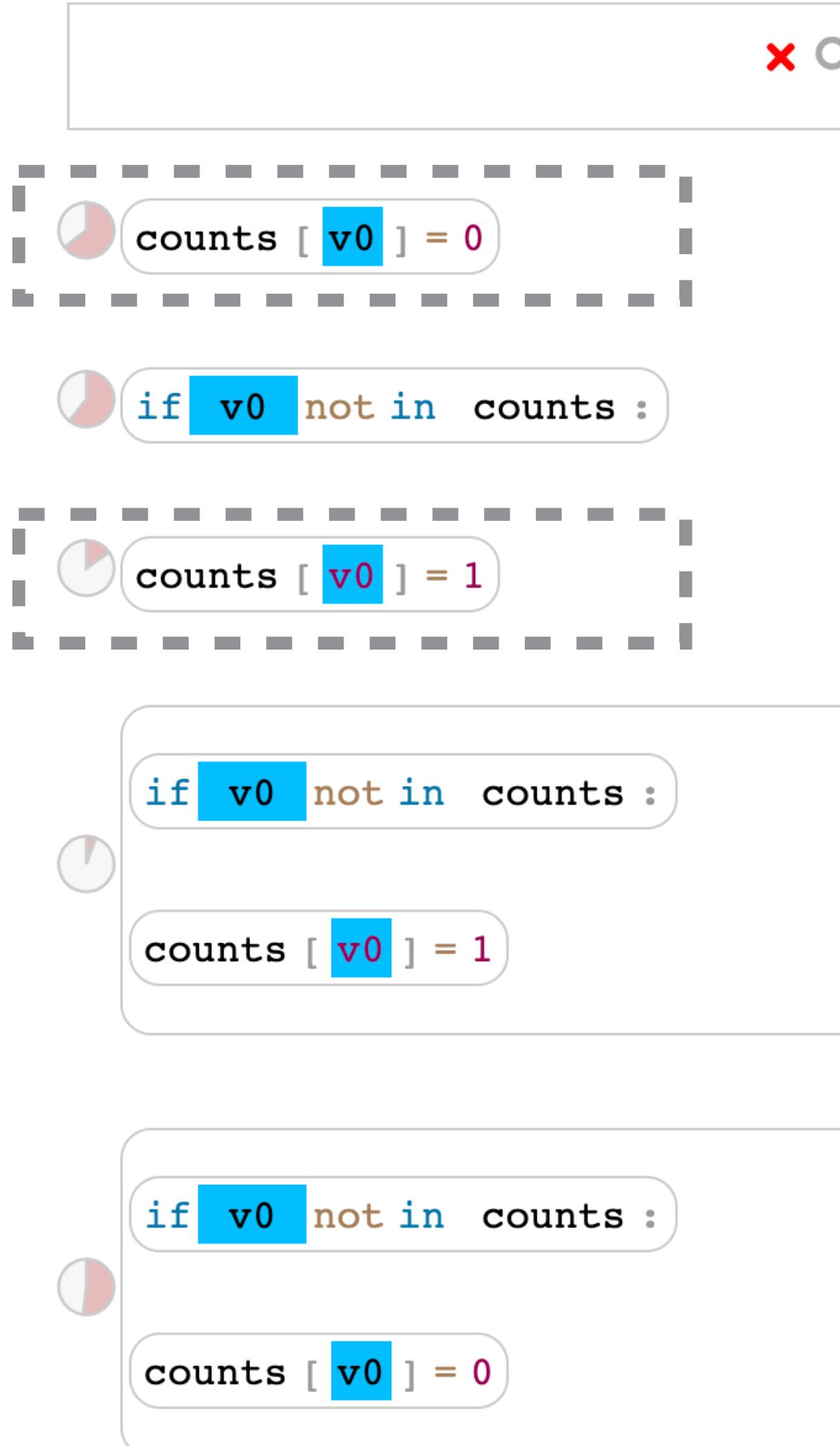
s1 = "hello"
counts = {}
for s in s1:
 counts[s] = s1.count(s)

s1 = "hello"
counts = {}
for ch in s1:
 if ch not in counts: count
 counts[ch] += 1

s1 = "hello"
counts = {}
for letter in s1:
 if letter in counts:
 counts[letter] += 1
 else:

Chain Queries

Two ways of initializing values in dict



Correct Incorrect

```
s1 = "hello"
counts={}
for i in s1:
    counts[i]=0
for i in s1:
    counts[i]+=1
```

```
s1 = "hello"
counts = {}
for s in s1:
    counts[s] += s1.count(s)
```

KeyError: h on line 4

```
s1 = "hello"
counts = {}
for s in s1:
    counts[s] = s1.count(s)
```

```
s1 = "hello"
counts = {}
for ch in s1:
    if ch not in counts: count
    counts[ch] += 1
```

```
s1 = "hello"
counts = {}
for c in s1:
    counts[c] = counts.get(c,
```

```
s1 = "hello"
counts = {}
for letter in s1:
    if letter in counts:
        counts[letter] += 1
    else:
```

Chain Queries

Two different combinations by chaining queries together

```
x 🔎
counts[v0] = 0
if v0 not in counts:
counts[v0] = 1
```

Correct Incorrect

```
s1 = "hello"
counts={} 
for i in s1:
    counts[i]=0
for i in s1:
    counts[i]+=1
```

```
s1 = "hello"
counts = {}
for s in s1:
    counts[s] += s1.count(s)
```

KeyError: h on line 4

```
s1 = "hello"
counts = {}
for s in s1:
    counts[s] = s1.count(s)
```

```
s1 = "hello"
counts = {}
for ch in s1:
    if ch not in counts: count
    counts[ch] += 1
```

```
s1 = "hello"
counts = {}
for c in s1:
    counts[c] = counts.get(c,
```

Chain Queries

Two different combinations by chaining queries together

The diagram illustrates two distinct ways to chain queries together, separated by a dashed line.

Top Path:

- Start with a pie icon and the query: `counts [v0] = 0`.
- Followed by an if block: `if v0 not in counts :`
- End with a pie icon and the query: `counts [v0] = 1`.

Bottom Path:

- Start with a pie icon and the query: `if v0 not in counts :`
- Followed by a pie icon and the query: `counts [v0] = 1`.
- End with a pie icon and the query: `counts [v0] = 0`.

Correct Incorrect

`s1 = "hello"`
`counts={}`
`for i in s1:`
 `counts[i]=0`
`for i in s1:`
 `counts[i]+=1`

`s1 = "hello"`
`counts = {}`
`for s in s1:`
 `counts[s] += s1.count(s)`

KeyError: h on line 4

`s1 = "hello"`
`counts = {}`
`for s in s1:`
 `counts[s] = s1.count(s)`

`s1 = "hello"`
`counts = {}`
`for ch in s1:`
 `if ch not in counts: count`
 `counts[ch] += 1`

`s1 = "hello"`
`counts = {}`
`for letter in s1:`
 `if letter in counts:`
 `counts[letter] += 1`
 `else:`

Chain Queries

Two different combinations by chaining queries together

The screenshot shows the RunEx interface with two code snippets and their results.

Code Snippet 1:

```

x Q
Correct ✓ Incorrect ✓

counts[v0] = 0
if v0 not in counts:
counts[v0] = 1

```

Code Snippet 2:

```

s1 = "hello"
counts={} for i in s1:
counts[i]=0 for i in s1:
counts[i]+=1

```

Result for Snippet 2:

Correct ✓ Incorrect ✓

s1 = "hello"
counts={} for s in s1:
counts[s] += s1.count(s)

KeyError: h on line 4

Code Snippet 3:

```

s1 = "hello"
counts={} for i in s1:
if i not in counts:
counts[i]=1
counts[i] = counts[i] +1
print(counts)

```

Not pass unit test

Code Snippet 4:

```

counts = {}
for c in s1:
counts[c] = counts.get(c,
else:

```

Participants

Within-subjects study



12 participants

10 have python teaching experience

2 are senior students that are experienced with Python

One 70-minute lab session, where participants answered quiz questions on students' mistakes and patterns using different systems

Baselines

Evaluate two aspects:

- The effectiveness of searching by **runtime values**
- A **user interface** that allows users to easily create search queries

Baselines

Evaluate two aspects:

- The effectiveness of searching by **runtime values**
- A **user interface** that allows users to easily create search queries

Baselines

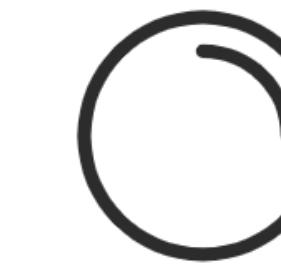
Evaluate two aspects:

- The effectiveness of searching by **runtime values**
- A **user interface** that allows users to easily create search queries

Two baseline systems



Limited user interface



Limited search capabilities

Baselines

Evaluate two aspects:

- The effectiveness of searching by **runtime values**
- A **user interface** that allows users to easily create search queries

	RunEx	Baseline 1	Baseline 2
Runtime values	✓	✗	✗
Text matching	✓	✓	✓
RunEx's UI			

Baselines

Evaluate two aspects:

- The effectiveness of searching by **runtime values**
- A **user interface** that allows users to easily create search queries

	RunEx	Baseline 1	Baseline 2
Runtime values	✓	✗	✗
Text matching	✓	✓	✓
RunEx's UI	✓	✗ “command+f” shortcut Type in regex expressions	✓

Results

Identify mistakes and patterns more accurately with runtime value search than the baselines

Accuracy of Quiz Questions

Pairs	M	SD	P
RunEx Baseline 1	0.892	0.193	<0.0001
	0.320	0.203	
RunEx Baseline 2	0.892	0.193	<0.001
	0.616	0.141	
Baseline 1 Baseline 2	0.320	0.203	<0.001
	0.616	0.141	

Results

Identify mistakes and patterns more accurately with runtime value search than the baselines

Accuracy of Quiz Questions

Pairs	M	SD	P
RunEx	0.892	0.193	
Baseline 1	0.320	0.203	<0.0001
RunEx	0.892	0.193	
Baseline 2	0.616	0.141	<0.001
Baseline 1	0.320	0.203	
Baseline 2	0.616	0.141	<0.001

Results

Identify mistakes and patterns more accurately with runtime value search than the baselines

*“... Having a runtime value (search), is almost like you have an **accessible debugger** where you can find that information... It feels very **accessible**, because you can match to different values, and you can set conditions on the values, which makes it **quite usable**. I’ve graded on other platforms where it’s really just text matching most of the times ...” (P10)*

Results

Identify mistakes and patterns more accurately with runtime value search than the baselines

The **quiz questions are realistic and representative** of information programming instructors would like to know

Results

Identify mistakes and patterns more accurately with runtime value search than the baselines

The **quiz questions are realistic and representative** of information programming instructors would like to know

Exact numbers of people that have a specific pattern or mistake

Whether **the majority or minority** of the class have similar patterns

Results

Identify mistakes and patterns more accurately with runtime value search than the baselines

The quiz questions are realistic and representative of information programming instructors would like to know

RunEx's user interface help participants **easily create search queries without prior knowledge**

Results

Identify mistakes and patterns more accurately with runtime value search than the baselines

The quiz questions are realistic and representative of information programming instructors would like to know

RunEx's user interface help participants **easily create search queries without prior knowledge**

Accuracy of Quiz Questions

Pairs	M	SD	P
Baseline 1	0.320	0.203	<0.001
Baseline 2	0.616	0.141	

Results

Identify mistakes and patterns more accurately with runtime value search than the baselines

The quiz questions are realistic and representative of information programming instructors would like to know

RunEx's user interface help participants **easily create search queries without prior knowledge**

	Baseline 1	Baseline 2
RunEx's UI	✗ “command+f” shortcut Type in regex expressions	✓

Google, Regex cheat sheets,
ChatGPT

Results

Identify mistakes and patterns more accurately with runtime value search than the baselines

The quiz questions are realistic and representative of information programming instructors would like to know

RunEx's user interface help participants **easily create search queries without prior knowledge**

	Baseline 1	Baseline 2
RunEx's UI	✗ “command+f” shortcut Type in regex expressions	✓

Google, Regex cheat sheets,
ChatGPT

Iterate and validate

Results

Identify mistakes and patterns more accurately with runtime value search than the baselines

The quiz questions are realistic and representative of information programming instructors would like to know

RunEx's user interface help participants **easily create search queries without prior knowledge**

	Baseline 1	Baseline 2
RunEx's UI	✗ “command+f” shortcut Type in regex expressions	✓

Google, Regex cheat sheets,
ChatGPT

Quickly create queries
without prior knowledge

Iterate and validate

Results

Identify mistakes and patterns more accurately with runtime value search than the baselines

The quiz questions are realistic and representative of information programming instructors would like to know

RunEx's user interface help participants easily create search queries without prior knowledge

RunEx **increases the expressiveness** of search queries

- Set maths on queries
- Chaining queries
- For patterns that appear similar at text level but have different runtime values, you can distinguish that with a simple query

Limitations

RunEx's implementation limits the types of runtime values searches possible.

```
A = B + 1
```

```
<<30>> = B + 1
```

```
__EVAL__(A, lambda v: v==30) = B + 1
```

Cannot execute this in Python

Limitations

RunEx's implementation limits the types of runtime values searches possible.

RunEx's implementation requires being able to run the code samples being searched - non-runnable code samples can only be searched through text matching

Limitations

RunEx's implementation limits the types of runtime values searches possible.

RunEx's implementation requires being able to run the code samples being searched - non-runnable code samples can only be searched through text matching

RunEx does not give users fine-grained control over how runtime values are searched

```
for i in List:  
    <>>
```

In every loop or in any loop?

Limitations

RunEx's implementation limits the types of runtime values searches possible.

RunEx's implementation requires being able to run the code samples being searched - non-runnable code samples can only be searched through text matching

RunEx does not give users fine-grained control over how runtime values are searched - in every loop or in any loop?

Code in our test dataset is relatively short and most are inexpensive to execute.

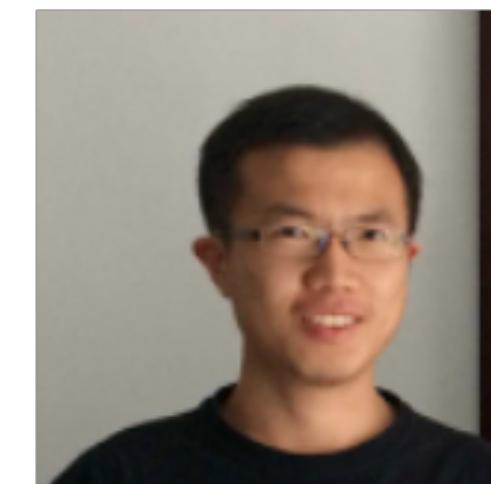
RunEx: Augmenting Regular-Expression Code Search with Runtime Values



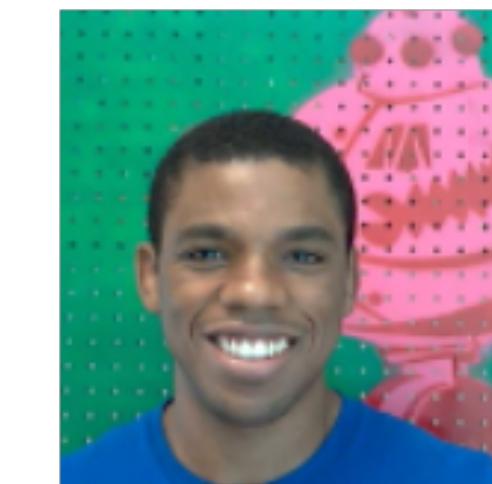
- A **syntax and mechanism that combine regular expressions and runtime value** specifications, enabling precise and targeted code search
- A **user interface designed for programming instructors** to easily create runtime and syntax-based queries and search through the resulting matches.
- A **user study** that compares the effectiveness of our proposed system with text-based code search.



Ashley Ge Zhang
University of Michigan



Yan Chen
Virginia Tech



Steve Oney
University of Michigan



<https://gezhangrp.com>

Scan to learn more about my work!