RunEx: Augmenting Regular-**Ex**pression Code Search with **Run**time Values

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

[1] University of Michigan
[2] Virginia Tech

Large Introductory Programming Courses

Large Introductory Programming Courses

Large Introductory Programming Courses

More than hundreds of students

Large Introductory Programming Courses

```
t", "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)
```

```
t", "bake", "beam", "confide", "gri
ist()

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

Large Introductory Programming Courses

- More than hundreds of students
- Viewing students' solutions is challenging

Large Introductory Programming Courses

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

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

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

1

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

11

```
words = ["adopt", "bake", "beam", "confide", "gri
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

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

```
words = ["adopt", "bake", "beam", "confide", "gri
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)
```

```
words = ["adopt", "bake", "beam", "confide", "gri
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)
```

```
words = ["adopt", "bake", "beam", "confide", "gri

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:
    …
```

Correct
90 students

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

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:
    …
```

Pattern taught in class

```
For key in dict.keys():
    …
```

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:
    …
```

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

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

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

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

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

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

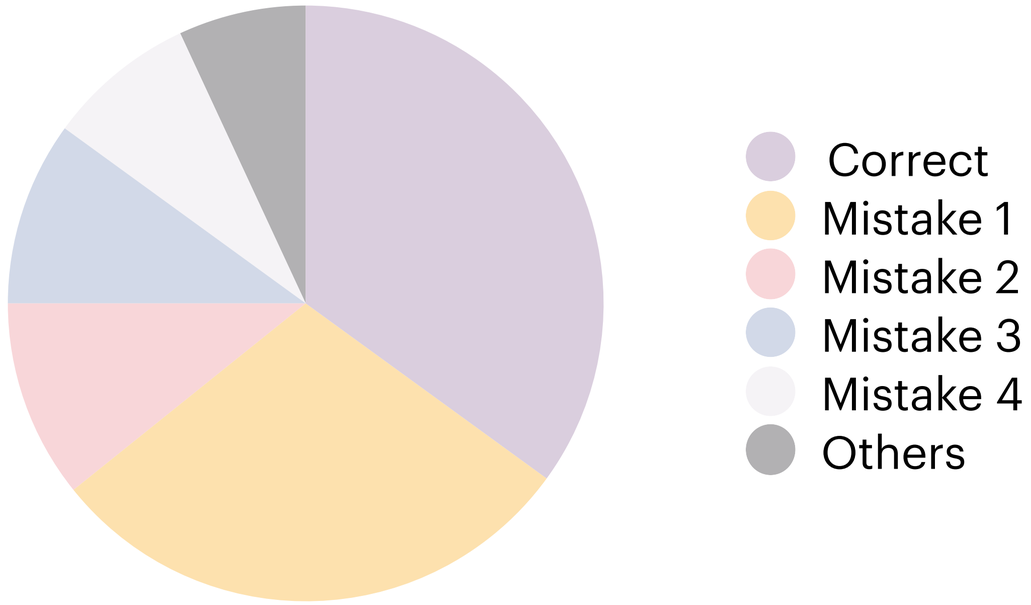
Instructors' Needs

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



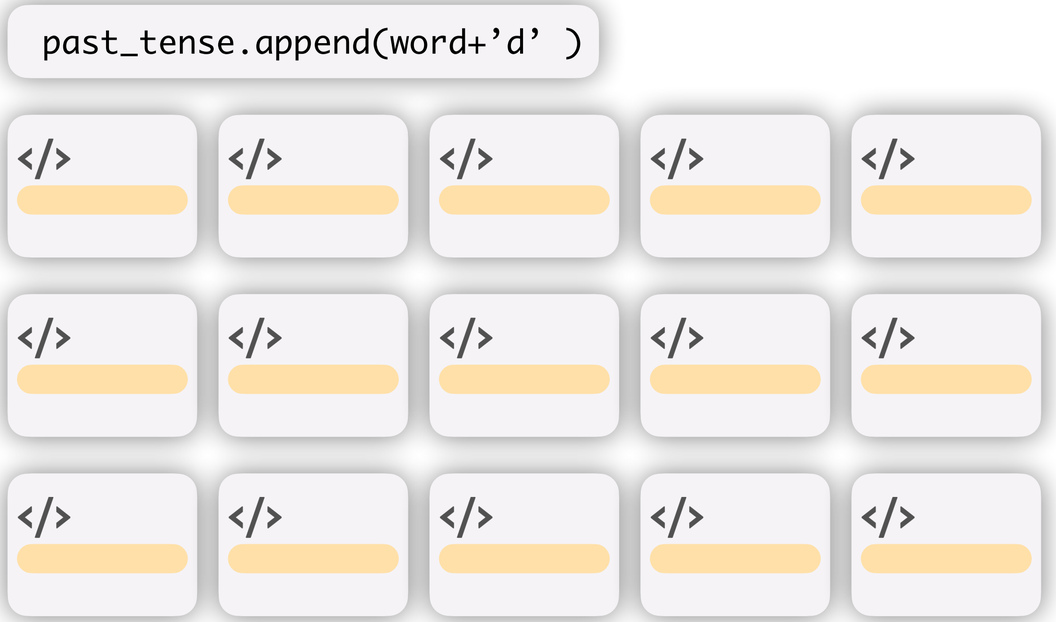
Instructors' Needs

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

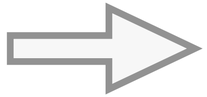
1. Find **common patterns and mistakes** in students' code
2. Get **descriptive statistics** about the class
3. Find **code samples that include similar code lines** to an existing pattern



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]
```

A list that contains the new word. e.g.
[“baked”], [“adapted”], [‘grilled’] …

Multiple correct ways

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']
```

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’] …

Text matching alone is not sufficient

Use Text Matching Alone is Insufficient

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

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

Use Text Matching Alone is Insufficient

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

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

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**

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

[4]Gu et al., Deep code search. ICSE 2018

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 + <<  >>
```

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", "gri
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", "gri
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", "gri
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", "gri

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

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

```
words = ["adopt", "bake", "beam", "confide", "gri
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)
```

```
words = ["adopt", "bake", "beam", "confide", "gri
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)
```

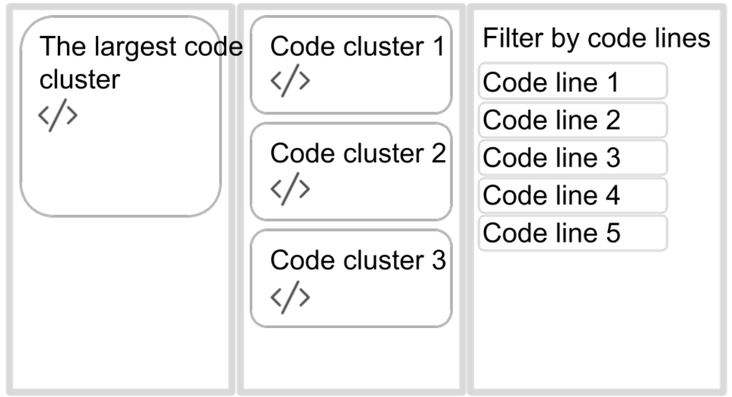
```
words = ["adopt", "bake", "beam", "confide", "gri

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**

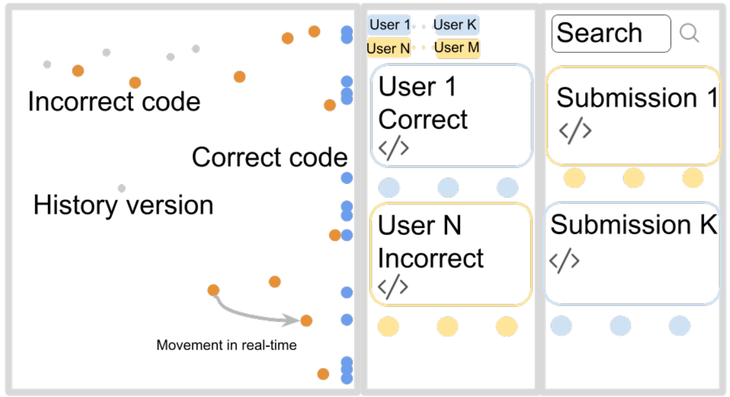
Execute the code samples to search across **runtime values**

Programming Education at Scale

OverCode[5]



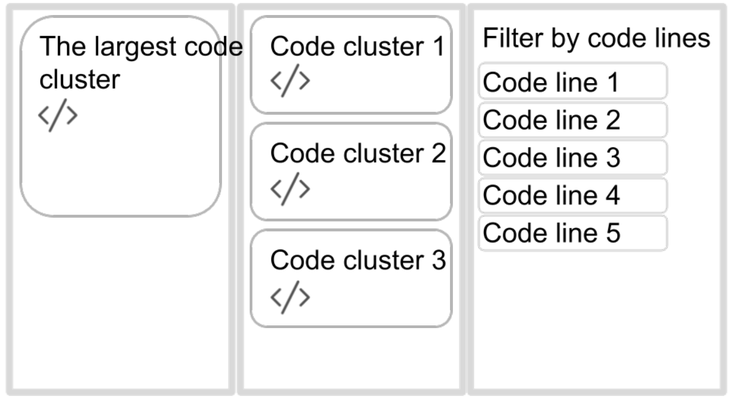
VizProg[6]



[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

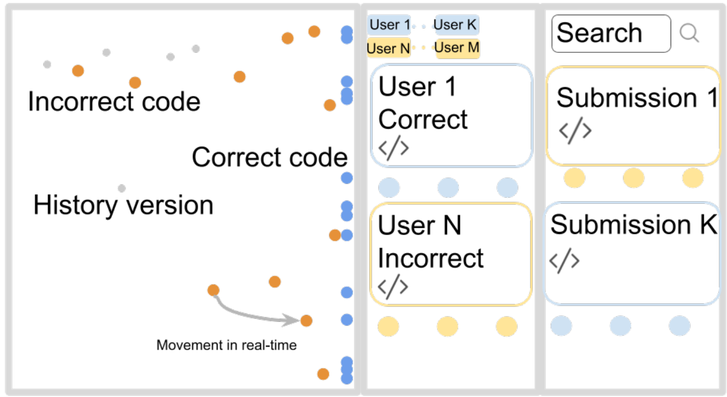
Programming Education at Scale

OverCode[5]



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

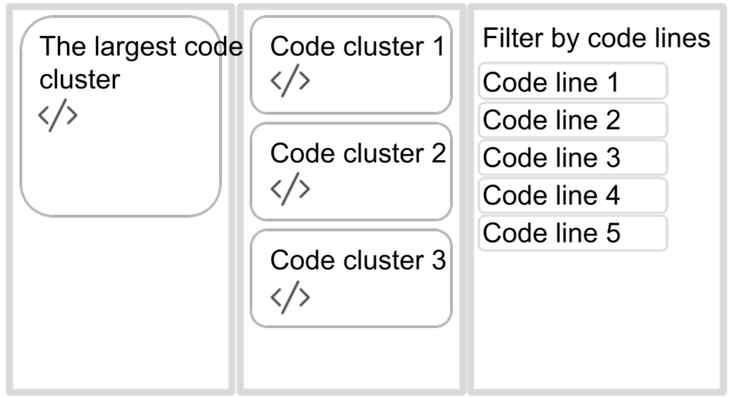
VizProg[6]



[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

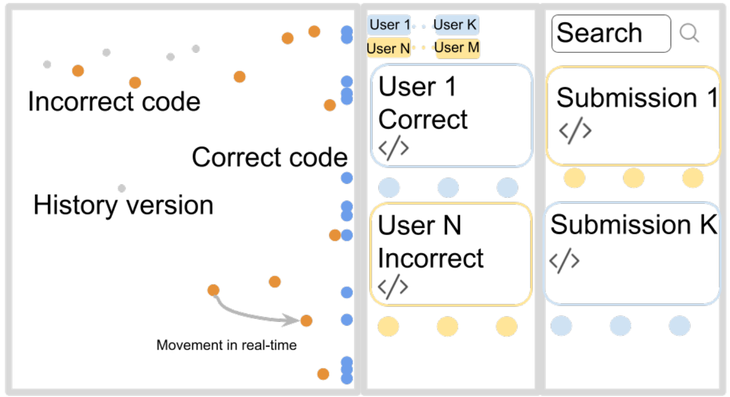
Programming Education at Scale

OverCode[5]



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

VizProg[6]

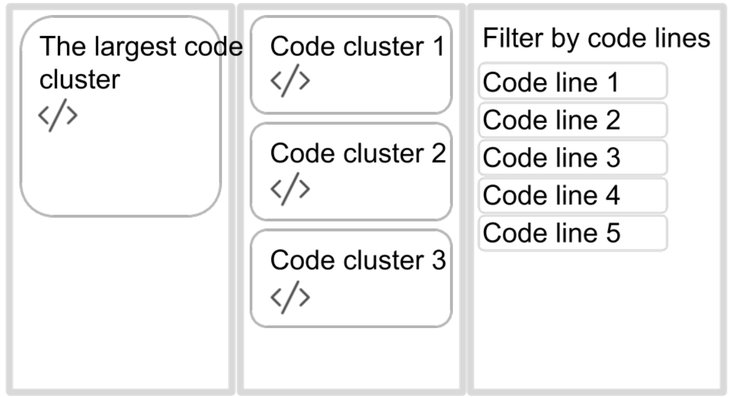


Uses **dynamic dots** to present students' coding progress on **a 2D map**

[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

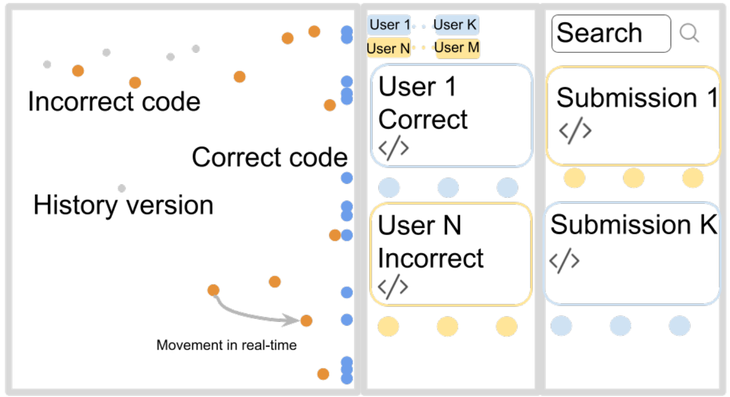
Programming Education at Scale

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**

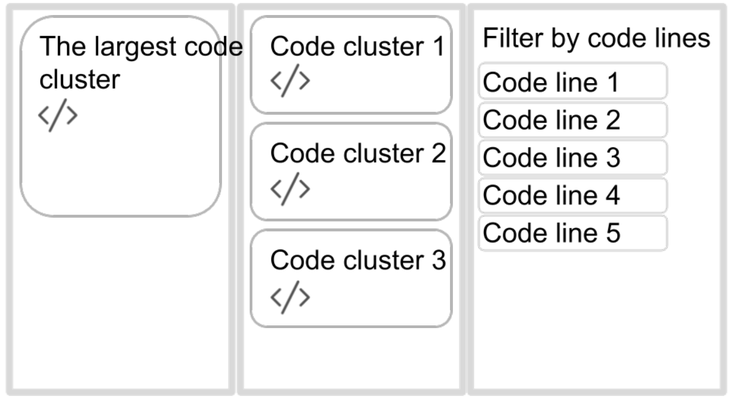
Instructors have **little control** over the results

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

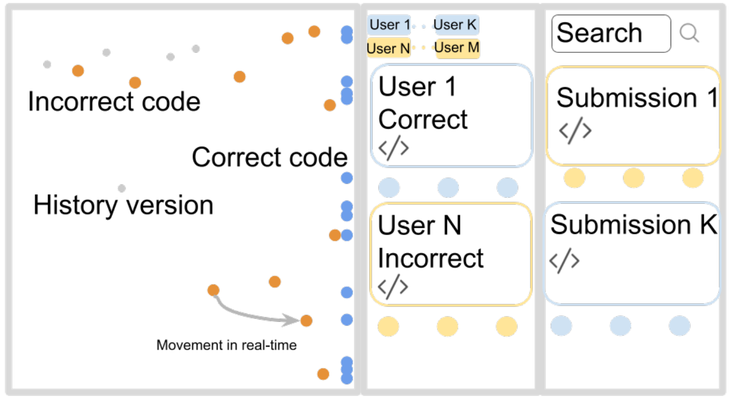
Programming Education at Scale

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

RunEx

1

Syntax

How does RunEx augment regular expression with runtime values?

1

Syntax

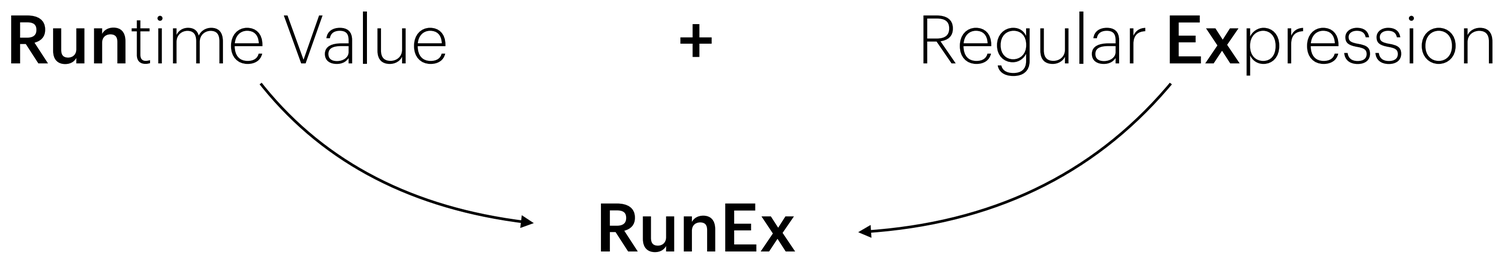
How does RunEx augment regular expression with runtime values?

User Interface

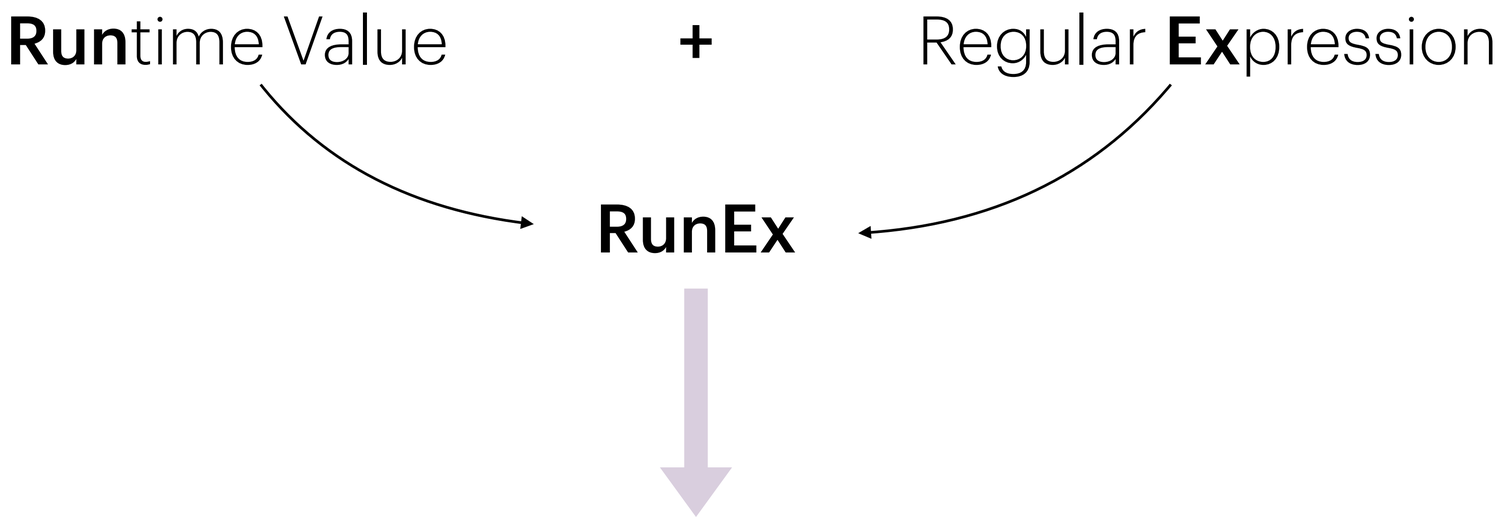
How do users create search queries in RunEx?

2

Syntax



Syntax



Greater expressiveness and precision
Maintain the flexibility of Regex

Syntax

Runtime Value

Constraints

<<...>>

Syntax

Runtime Value

Constraints

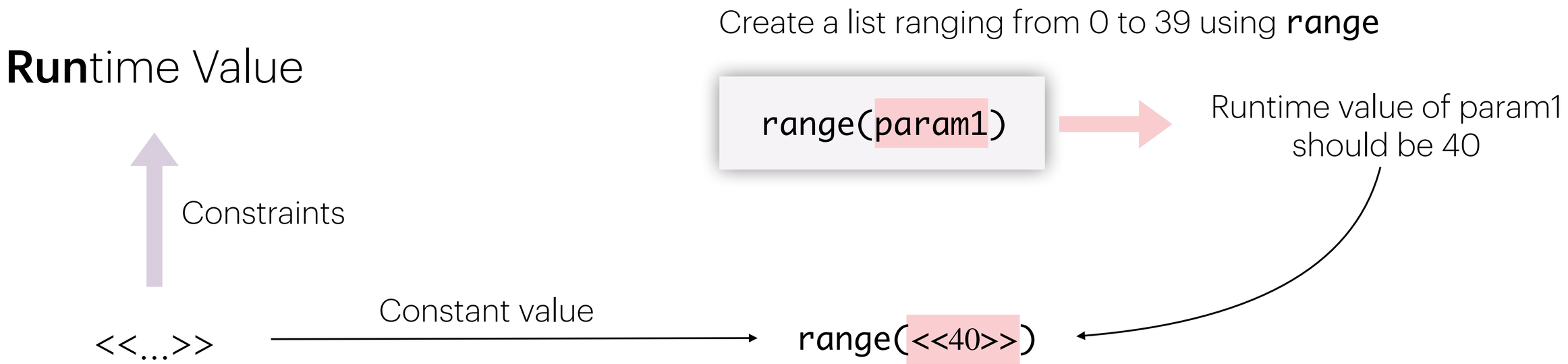
<<...>>

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

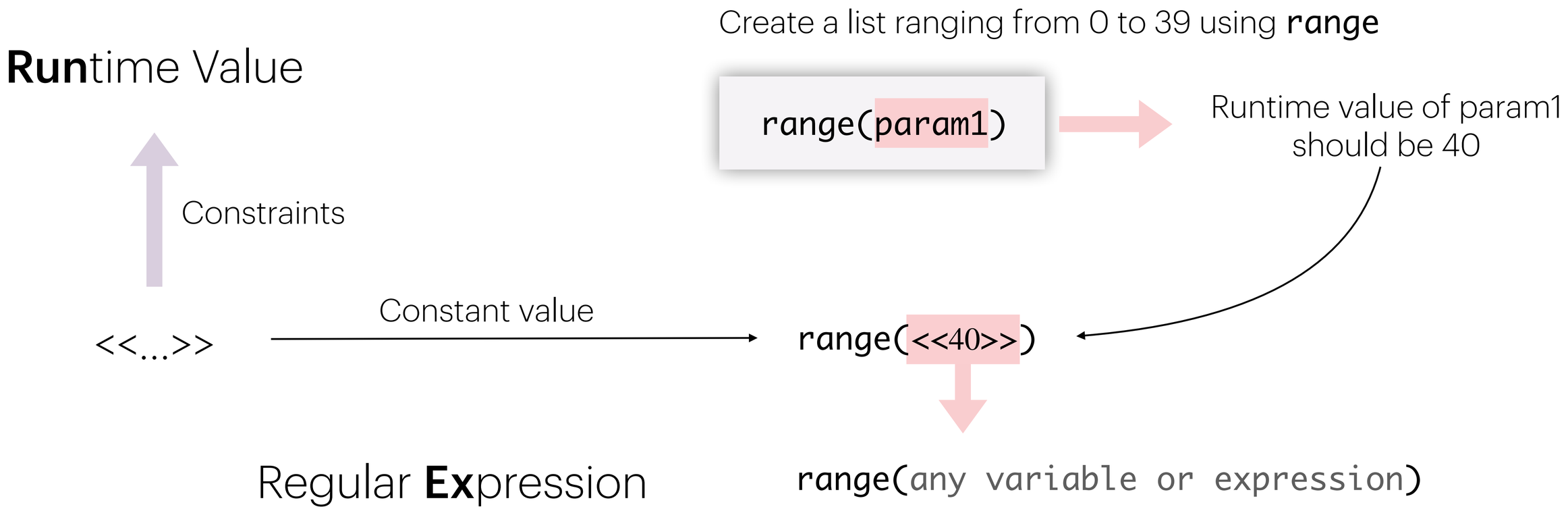
```
range(param1)
```


Runtime value of param1 should be 40

Syntax


Create a list ranging from 0 to 39 using range
Runtime Value
range(param1)
Runtime value of param1 should be 40
Constraints
Constant value
<<...>>
range(<<40>>)

Syntax

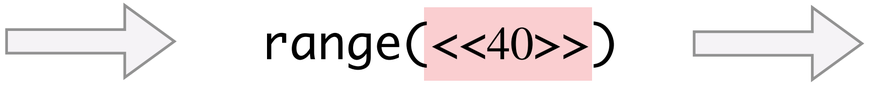

Create a list ranging from 0 to 39 using range
Runtime Value
range(param1)
Runtime value of param1 should be 40
Constraints
Constant value
<<...>>
range(<<40>>)
Regular Expression
range(any variable or expression)

Syntax

Runtime Value

+

Regular **Ex**pression




```
range(20+20)
```

```
K = 40
range(K)
```

```
range(40)
```

Syntax

Runtime Value

+

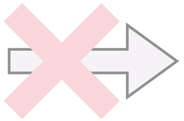
Regular **Ex**pression

`range(<<40>>)`

```
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

Constraints

Python conditions

<<...>>

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

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 + <<λ_v>>

```
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

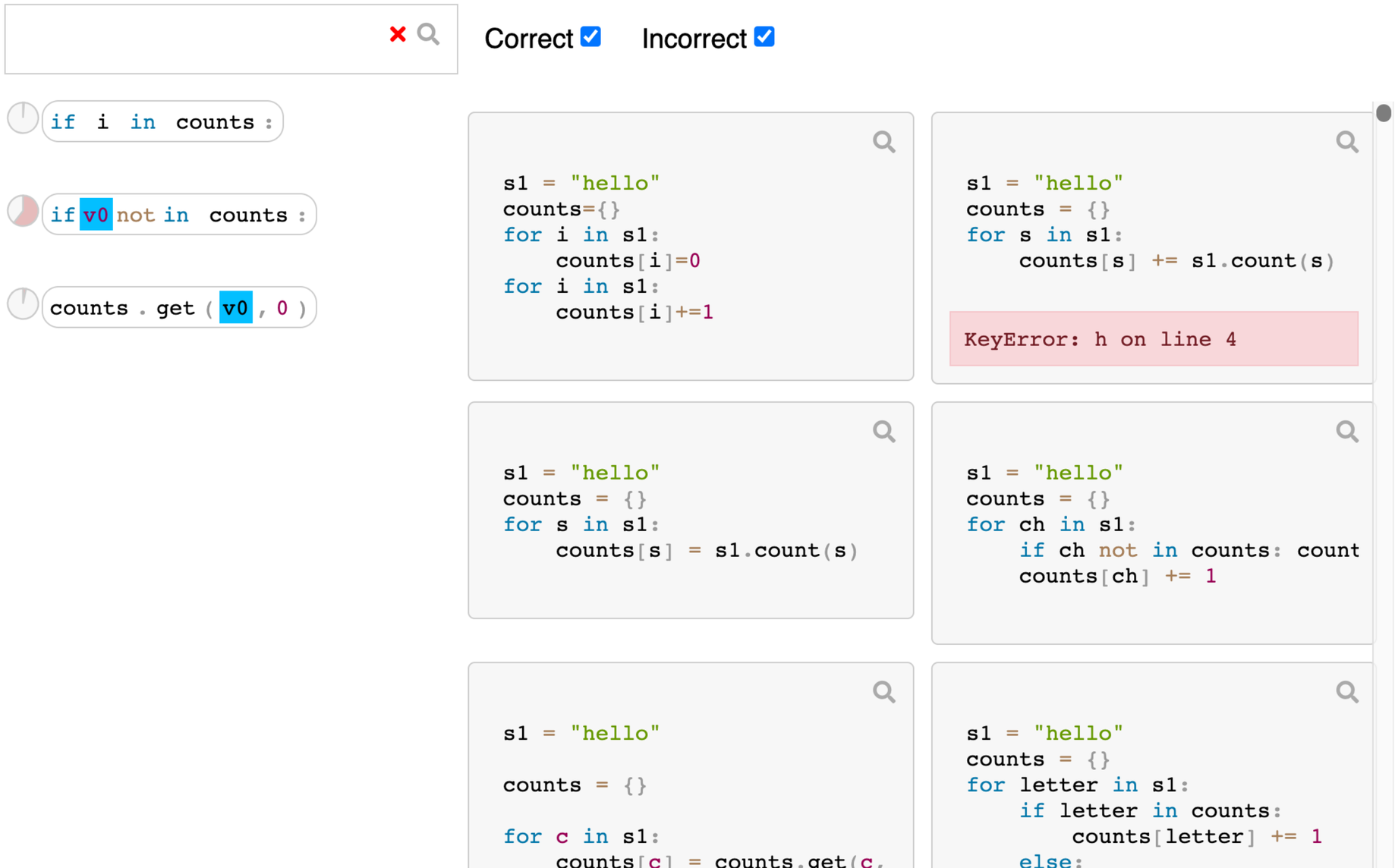
```
SIGNAL = False
def __EVAL__(variable, fn):
    global SIGNAL
    if fn(variable):
        SIGNAL = True
    return variable
```

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

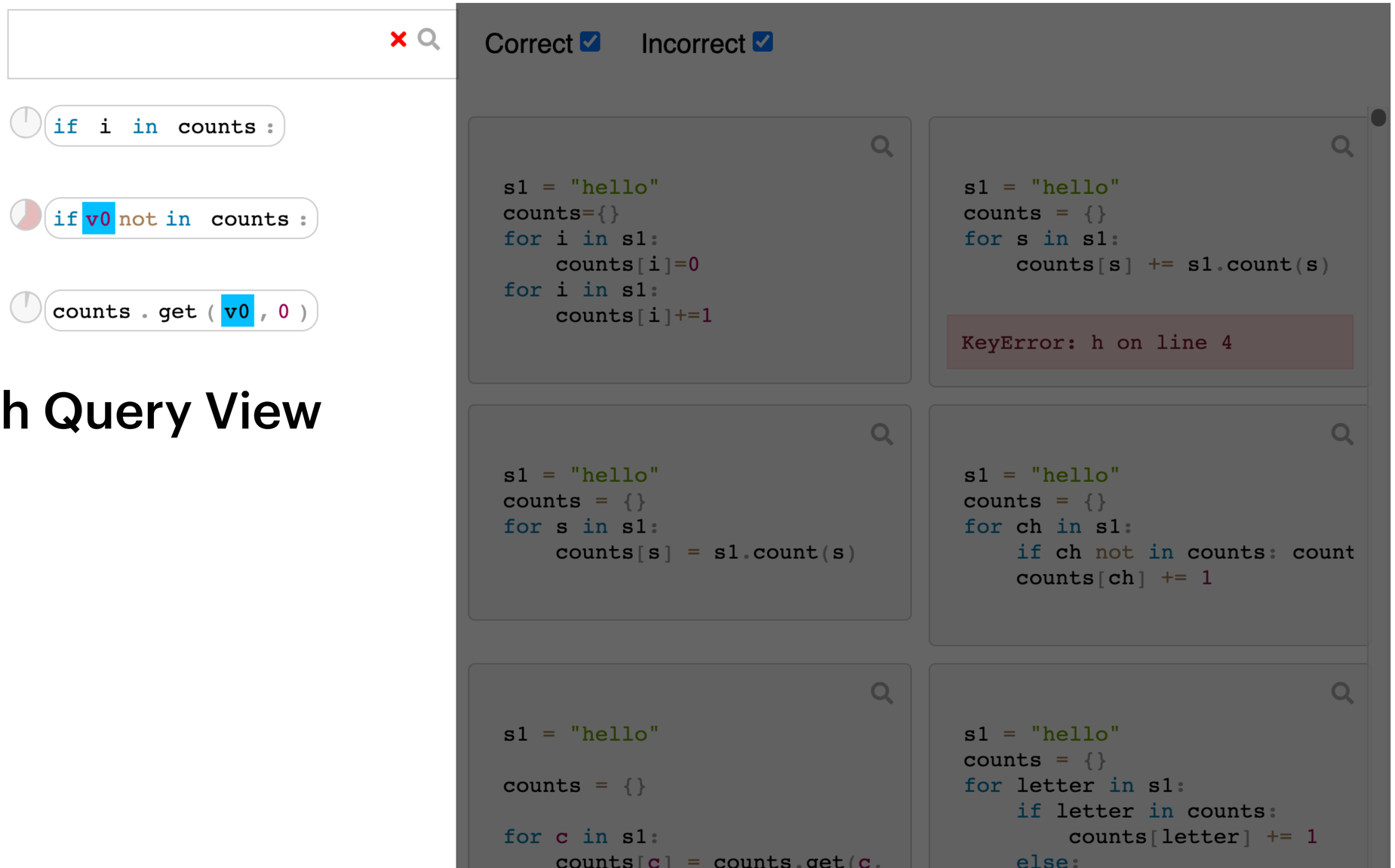
Inject into the original code

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

User Interface

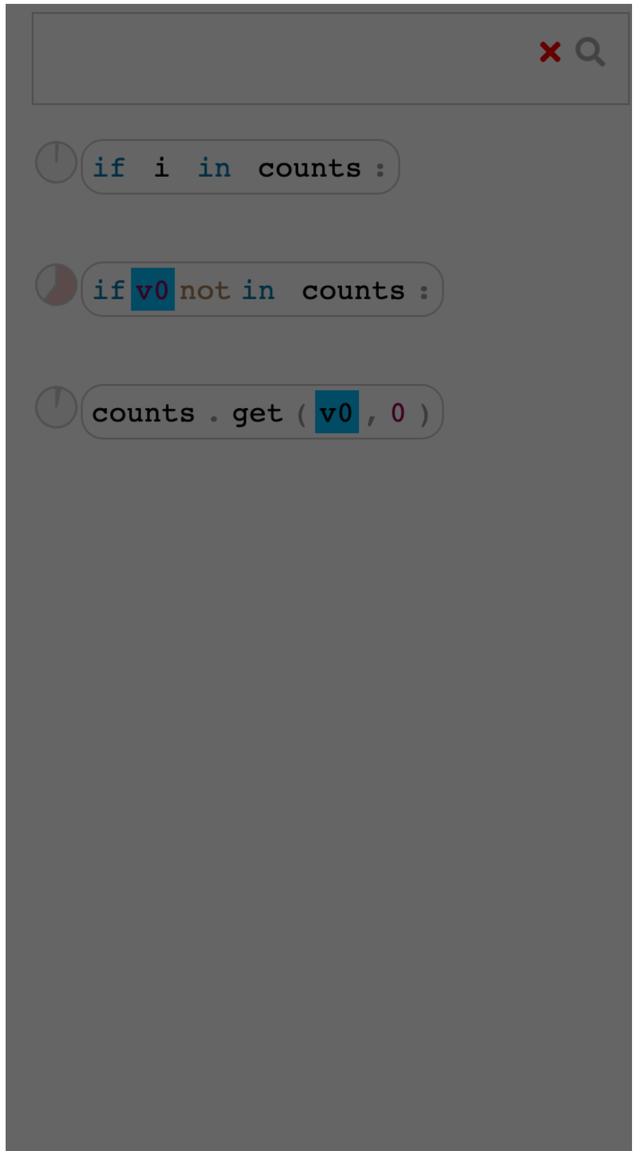


User Interface



Search Query View

User Interface



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
```

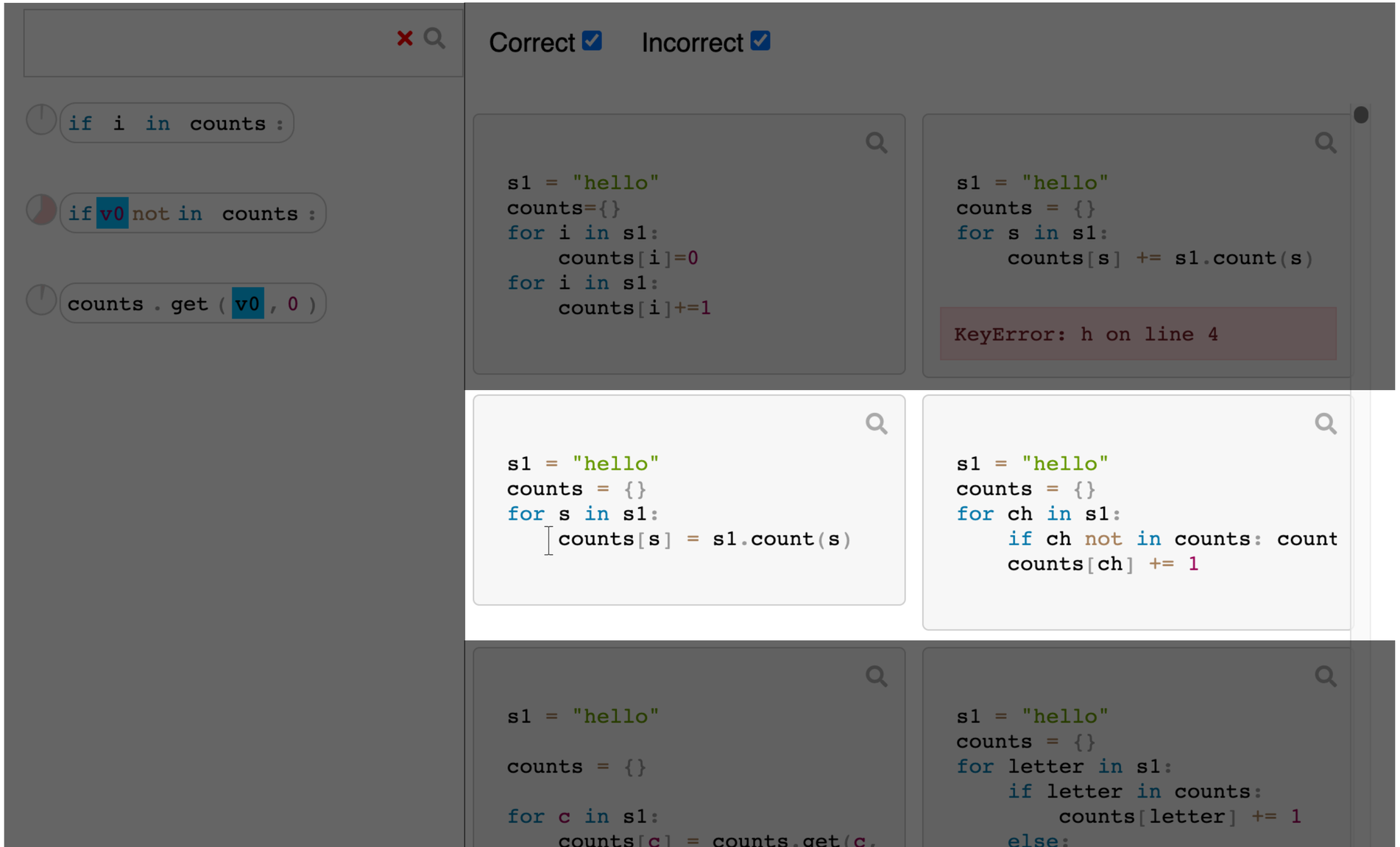
```
s1 = "hello"
counts = {}
for s in s1:
    counts[s] = s1.count(s)
```

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

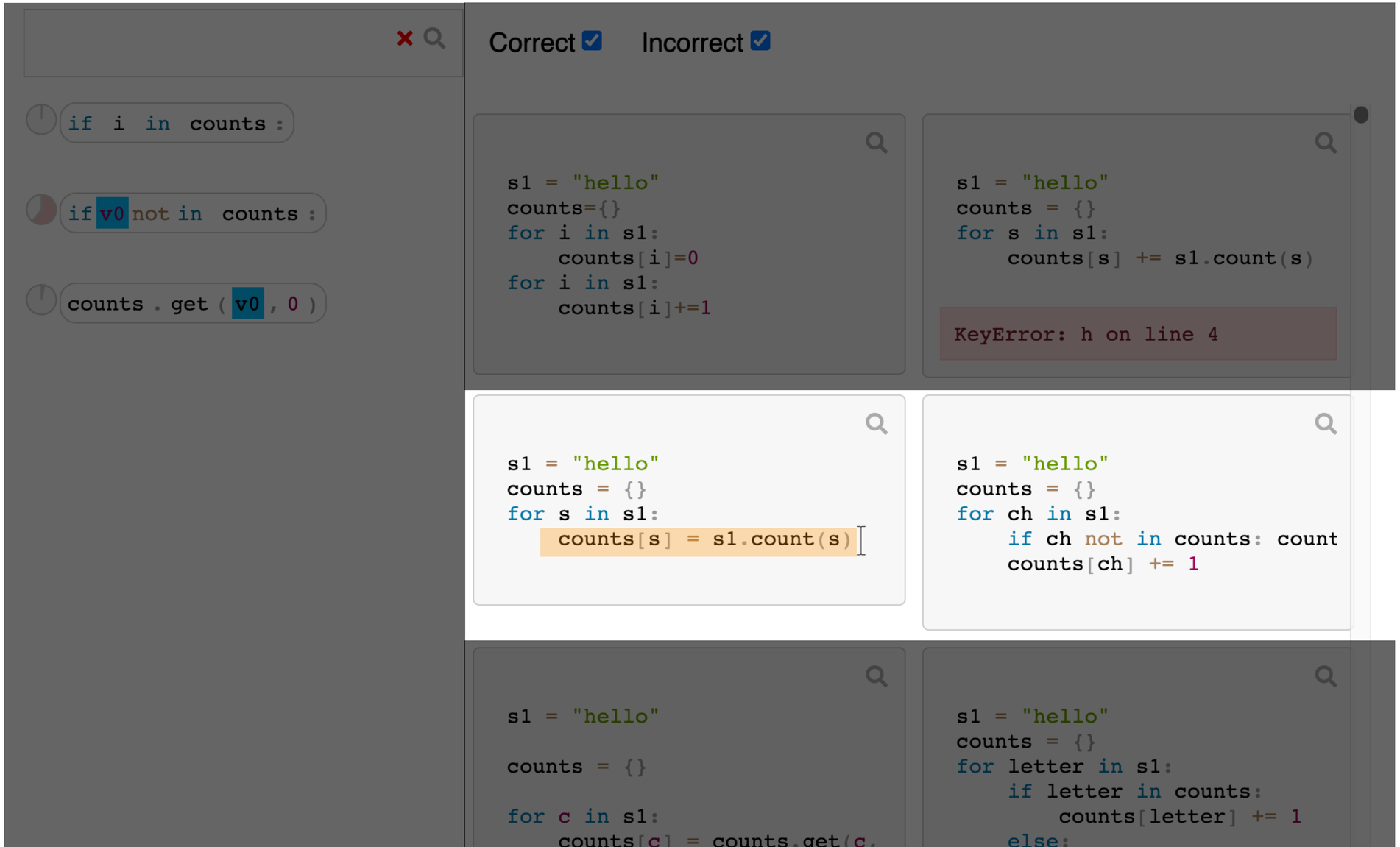
```
s1 = "hello"

counts = {}

for c in s1:
```

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

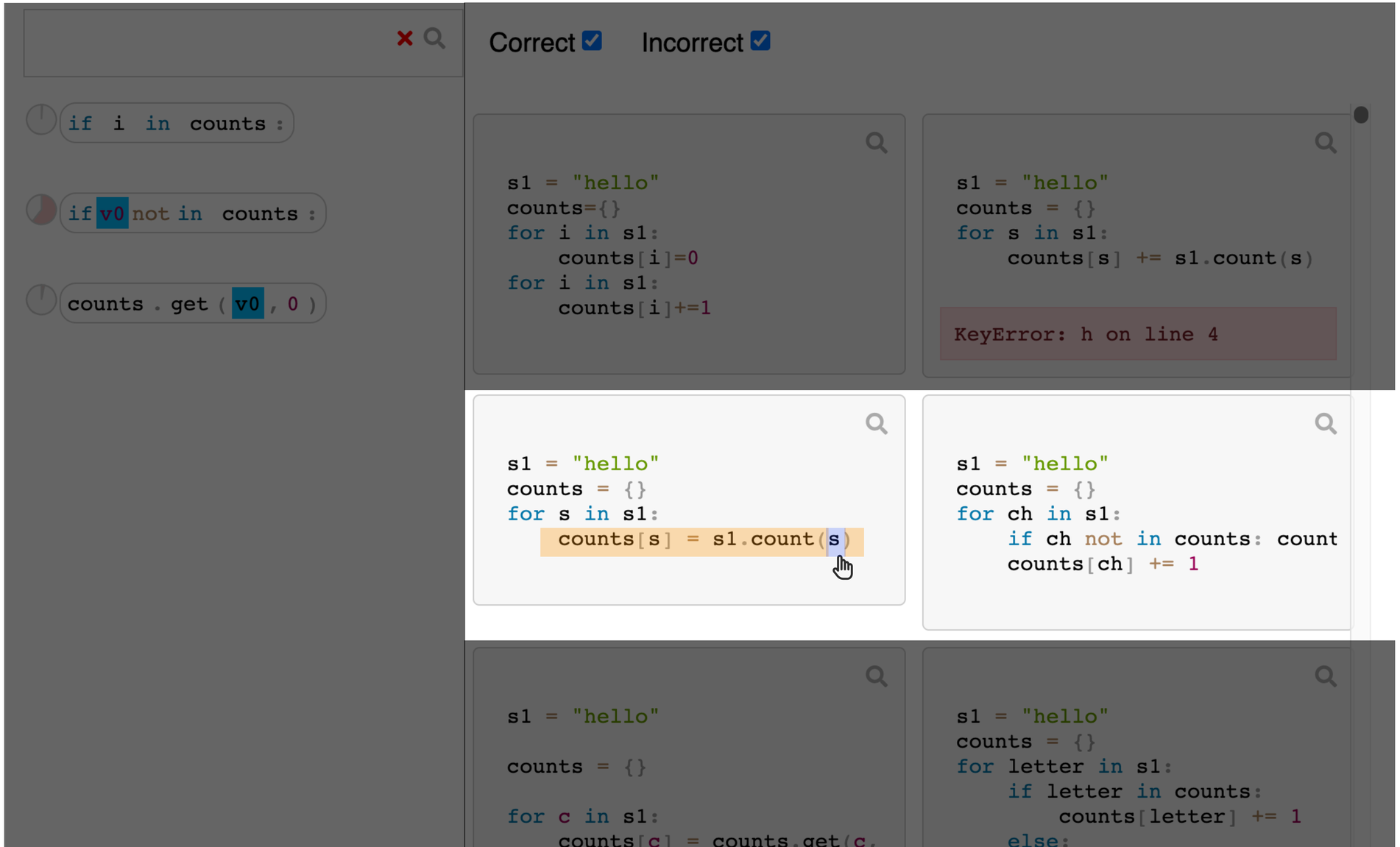
User Interface



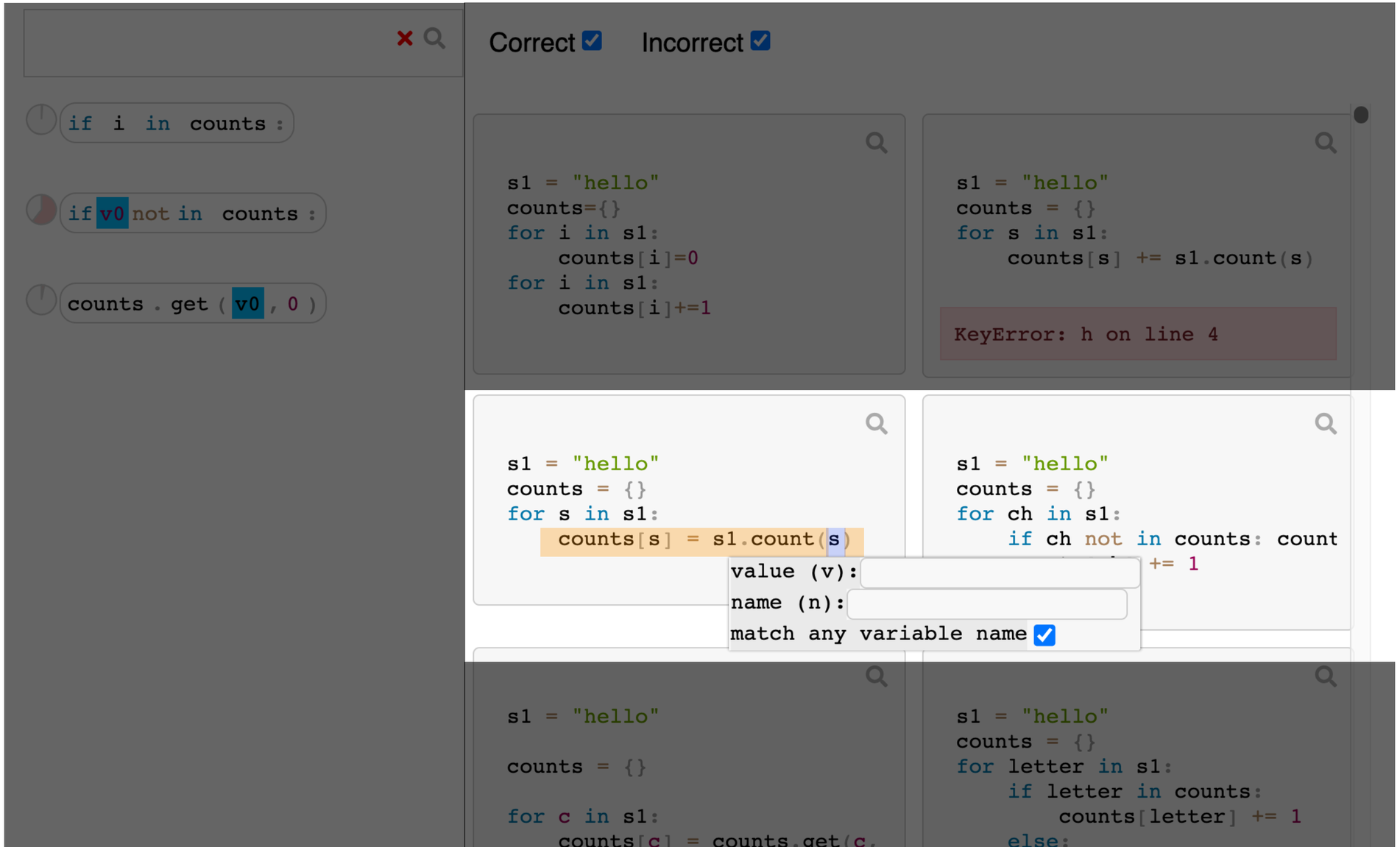
User Interface



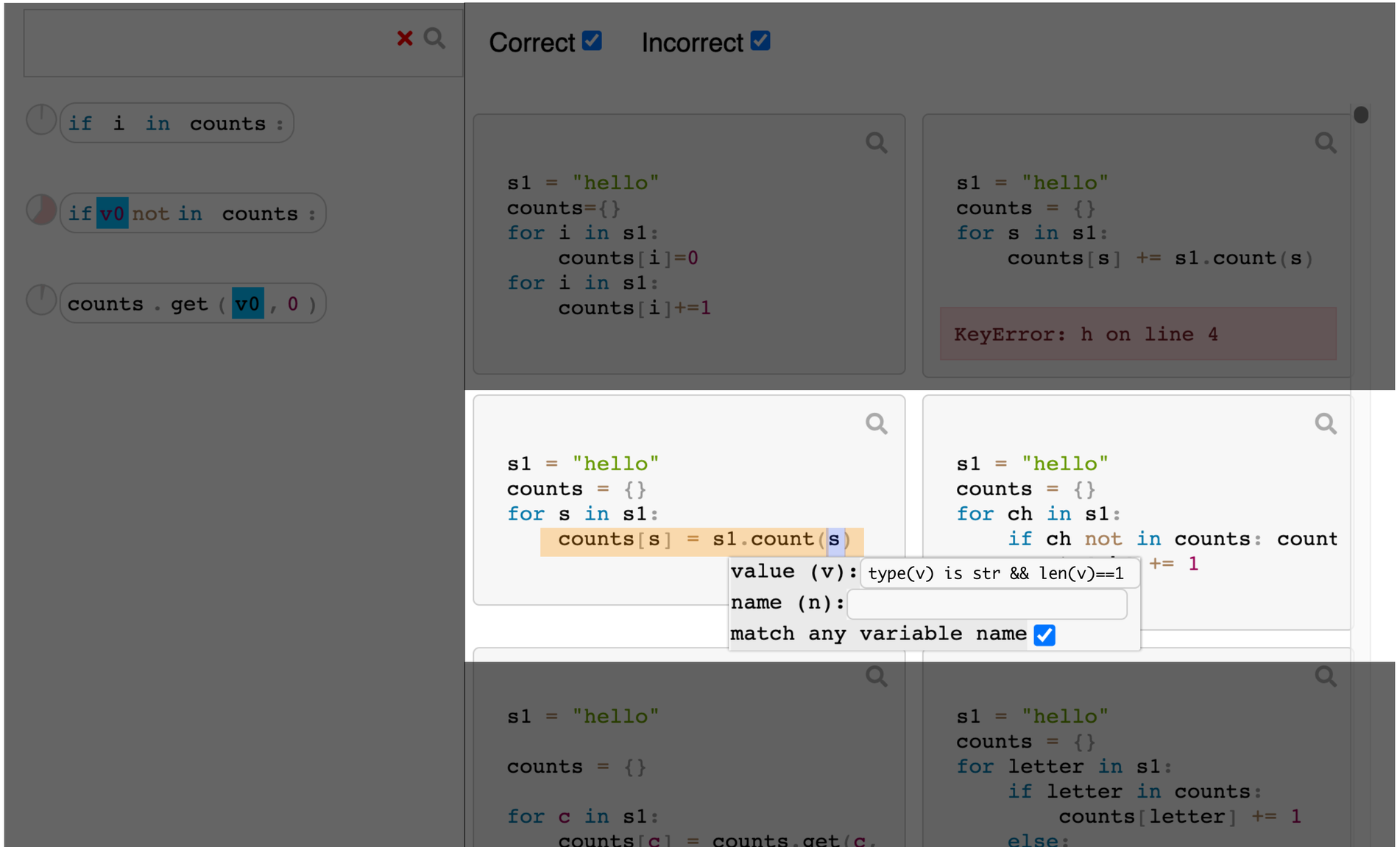
User Interface



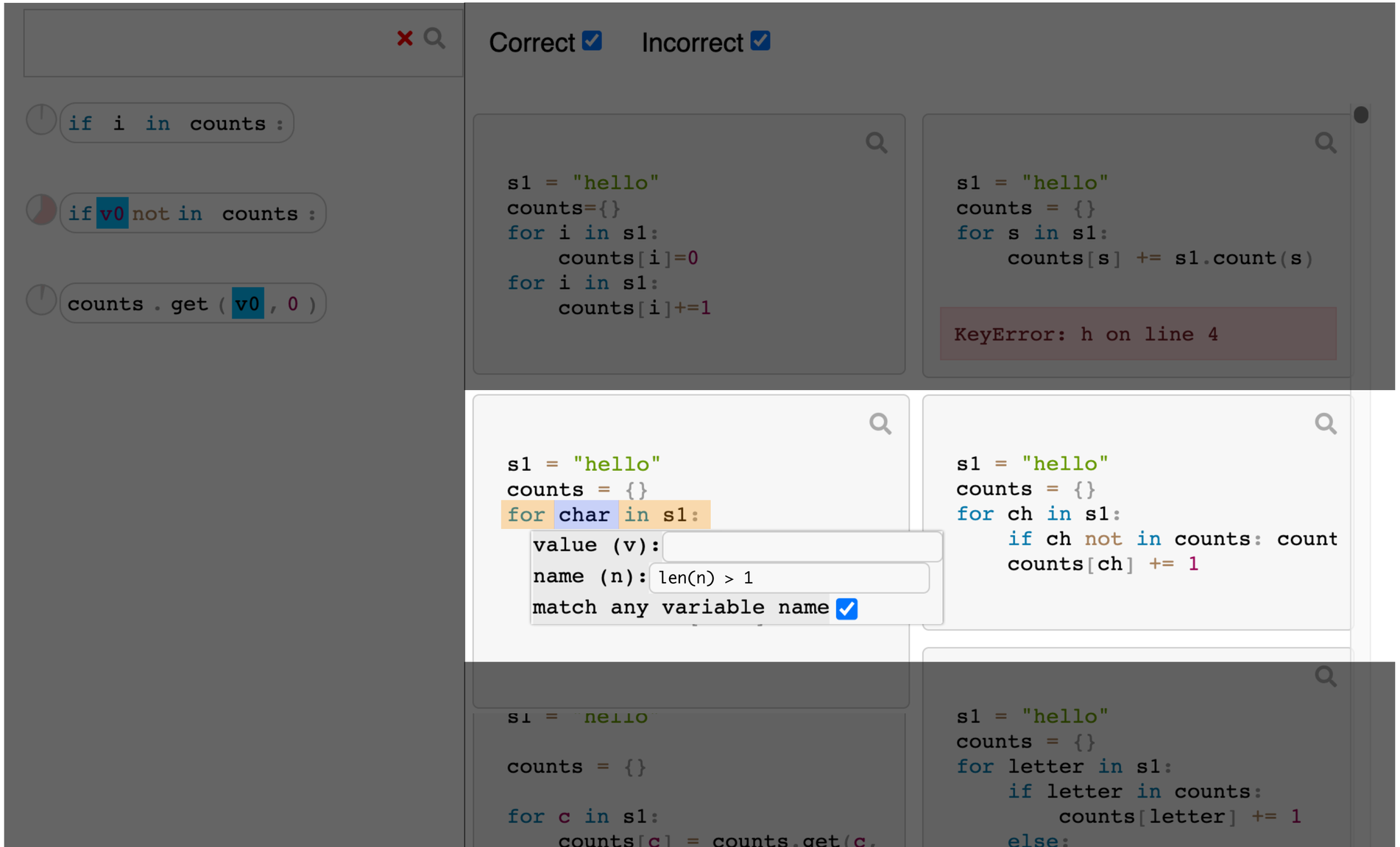
User Interface



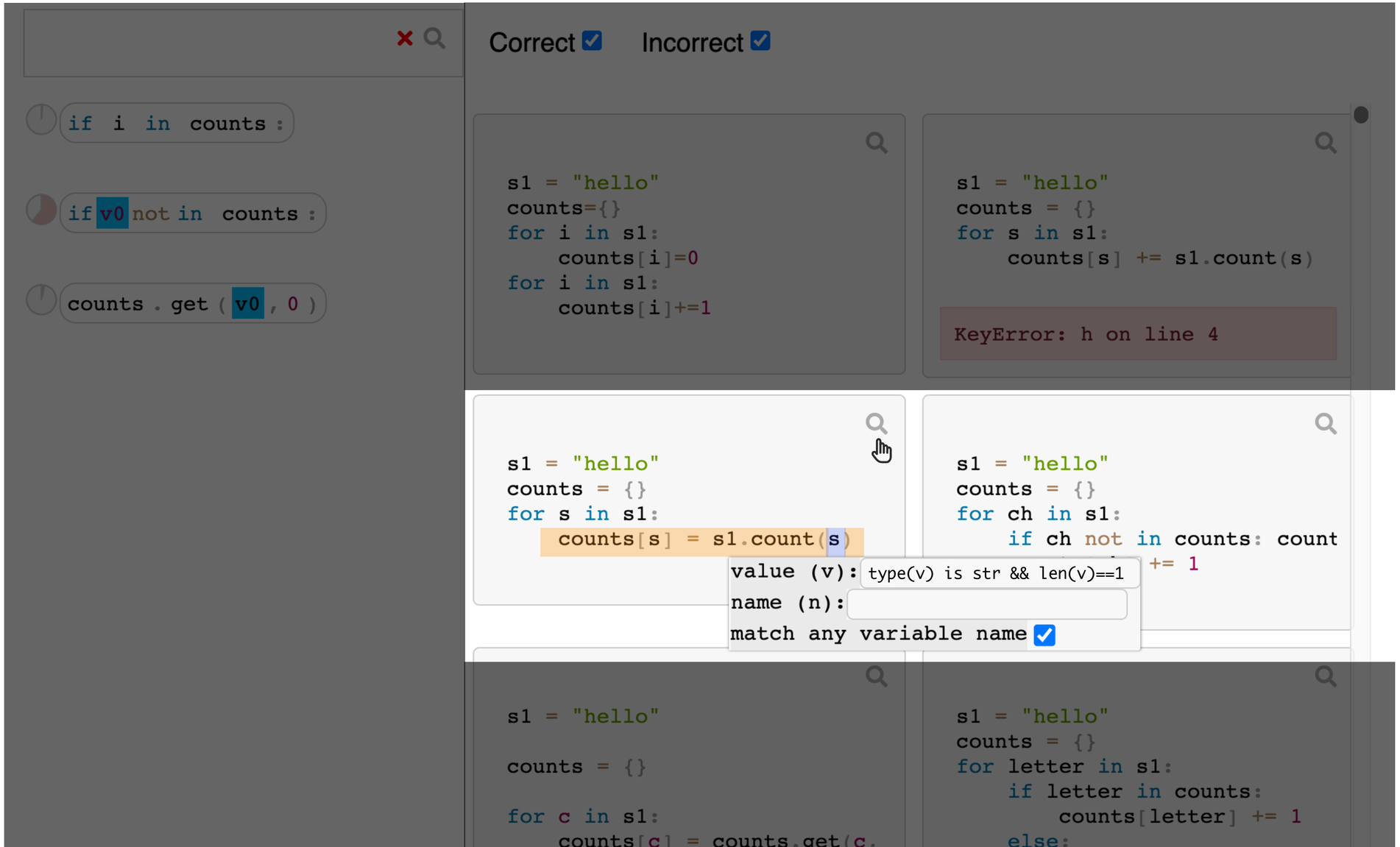
User Interface



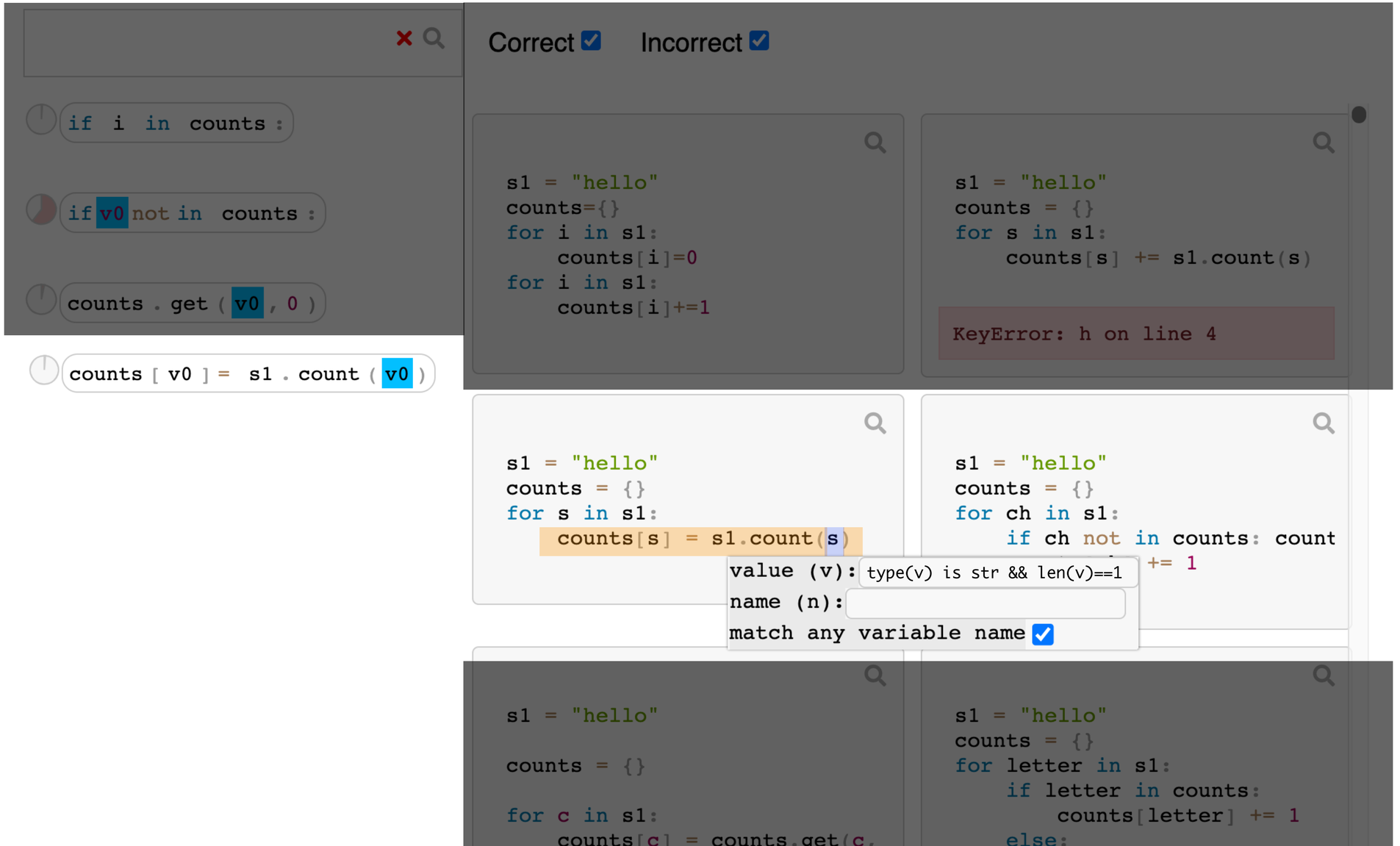
User Interface



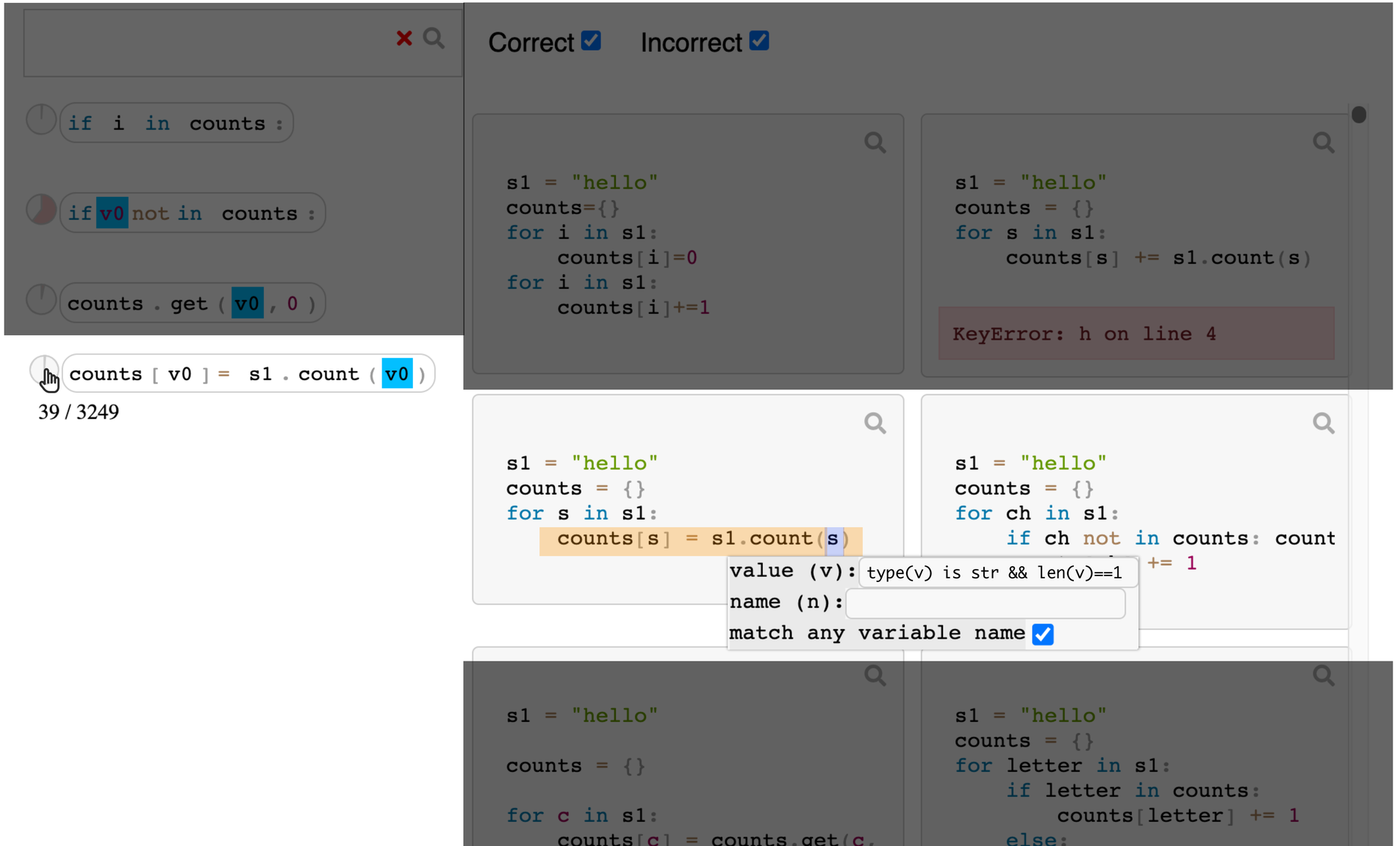
User Interface



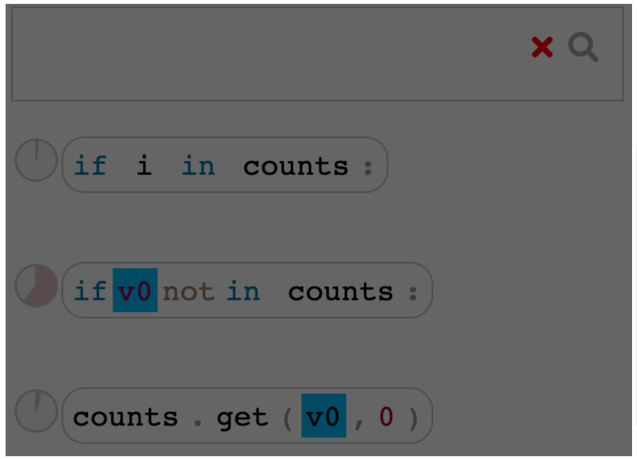
User Interface



User Interface



User Interface



Correct ☑ Incorrect ☑

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

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

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

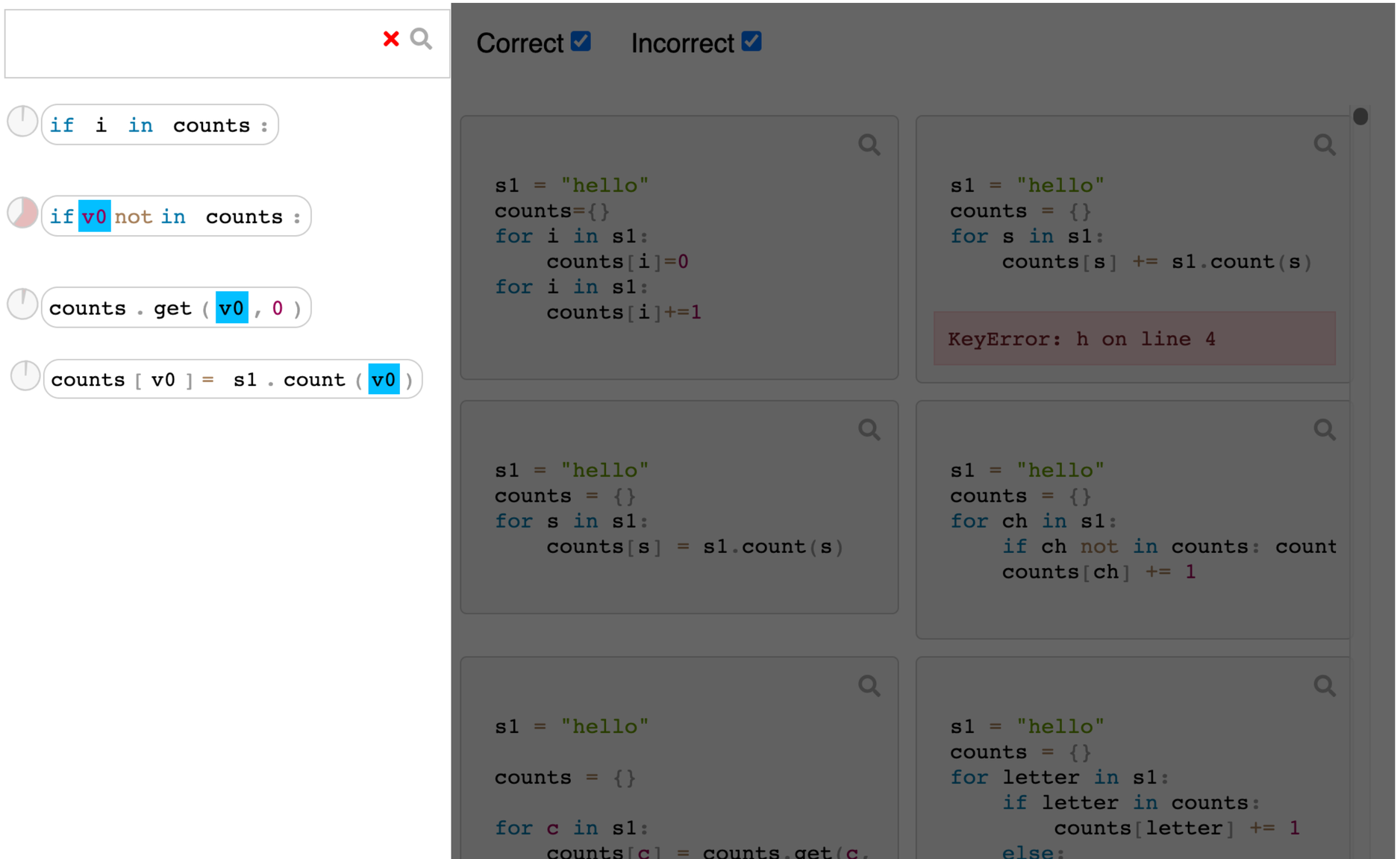
```
s1 = "hello"
counts = {}
for x in set(s1):
    counts[x] = s1.count(x)
```

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

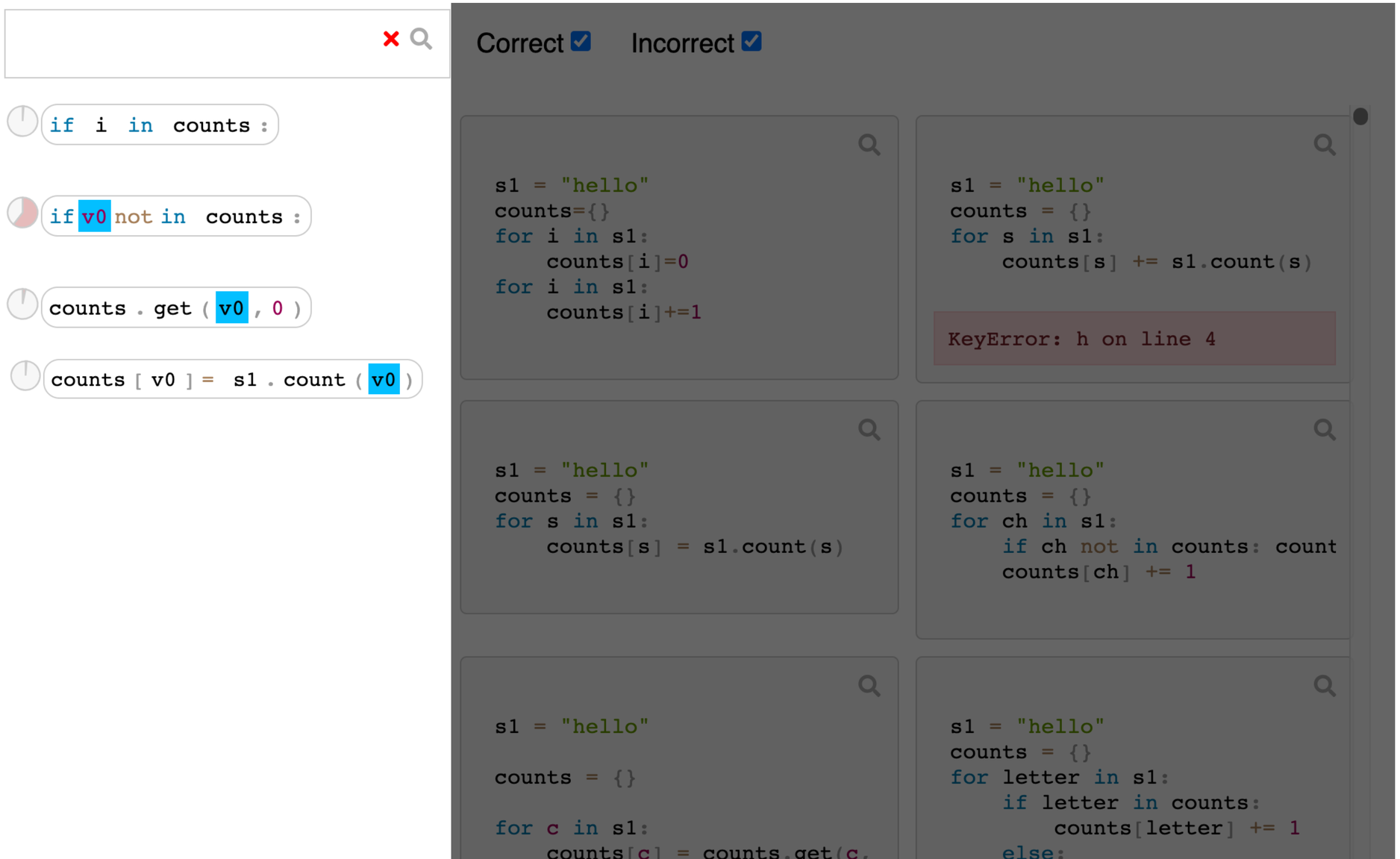
```
s1 = "hello"
counts = {}
for letter in s1:
    counts[letter] = s1.count(
print(freq)
```

```
NameError: name 'freq' is not
```

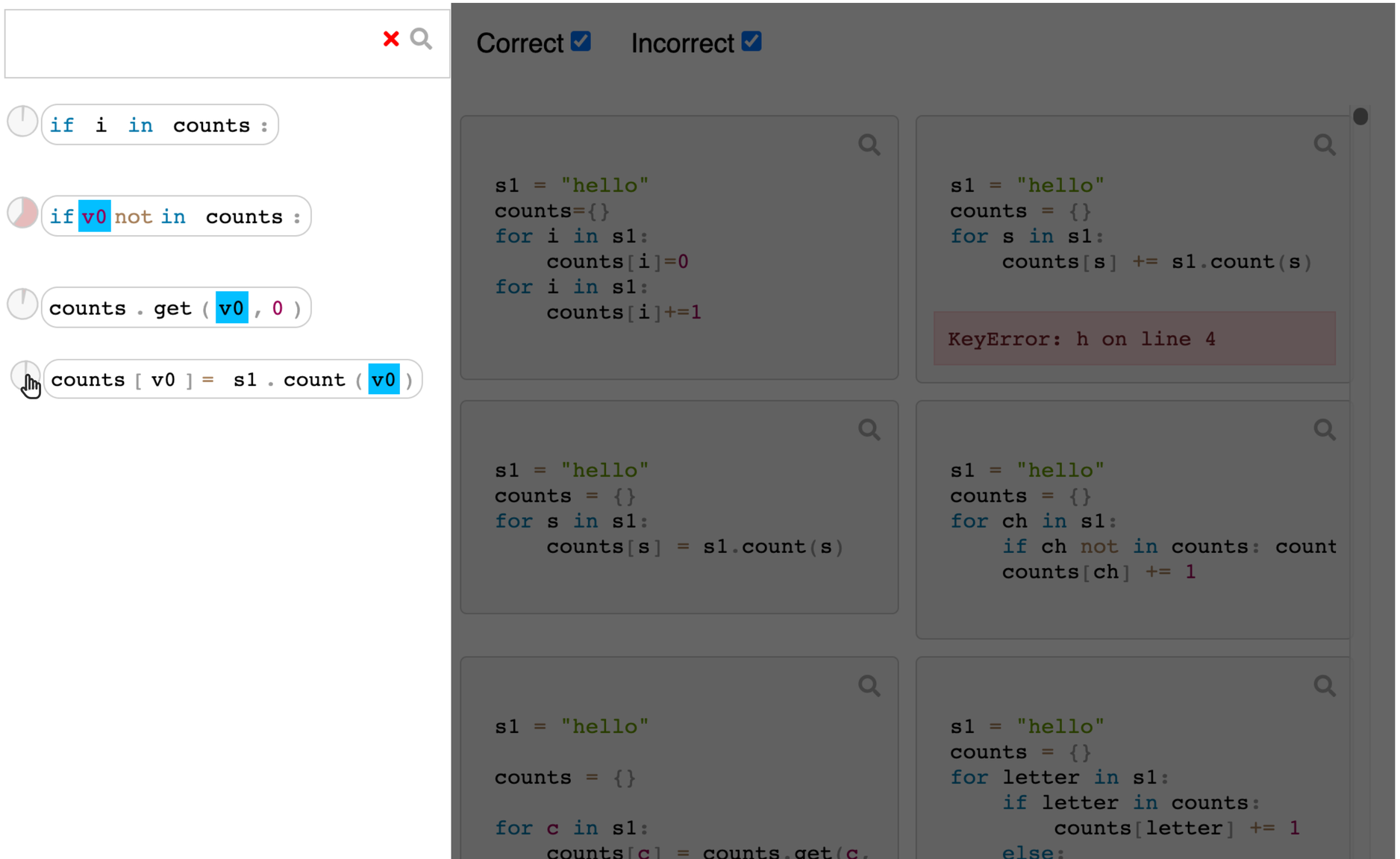
User Interface



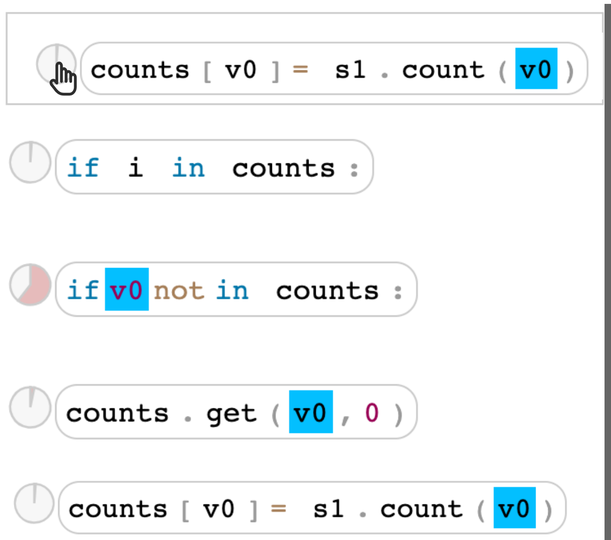
Set Maths on Queries

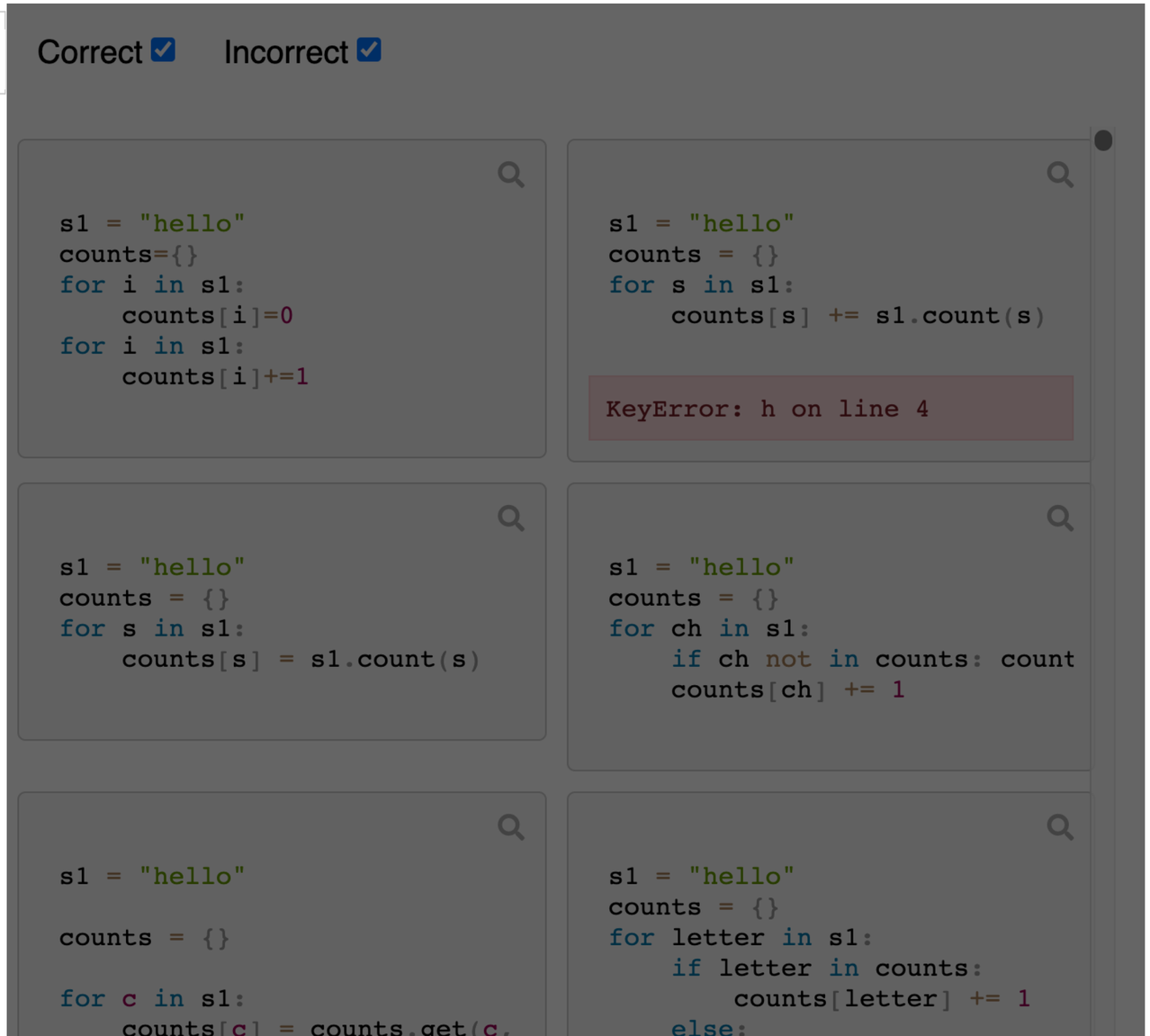


Set Maths on Queries



Set Maths on Queries





Set Maths on Queries

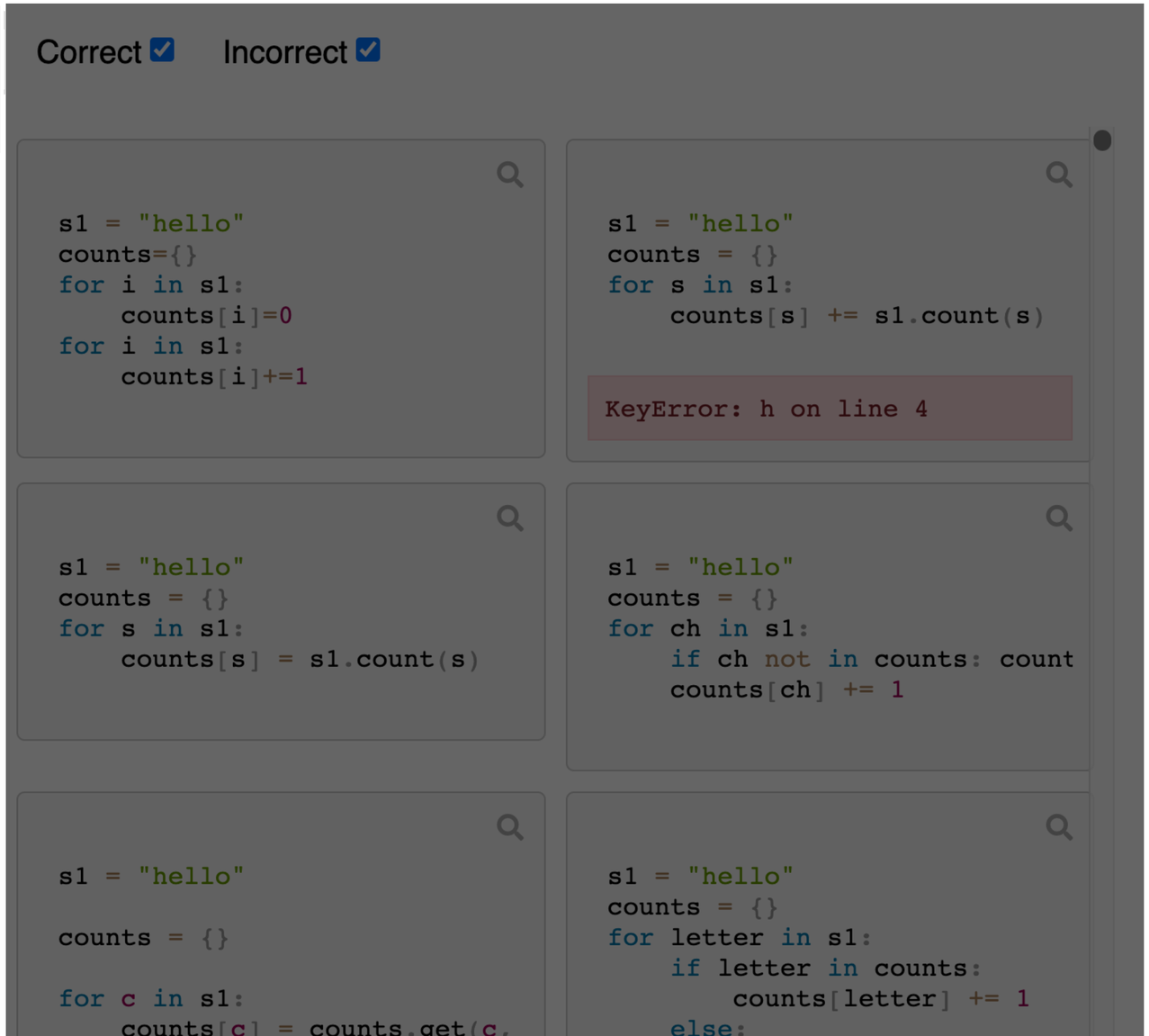
```
counts [ v0 ] =  s1 . count ( v0 )
```

```
if  i  in  counts :
```

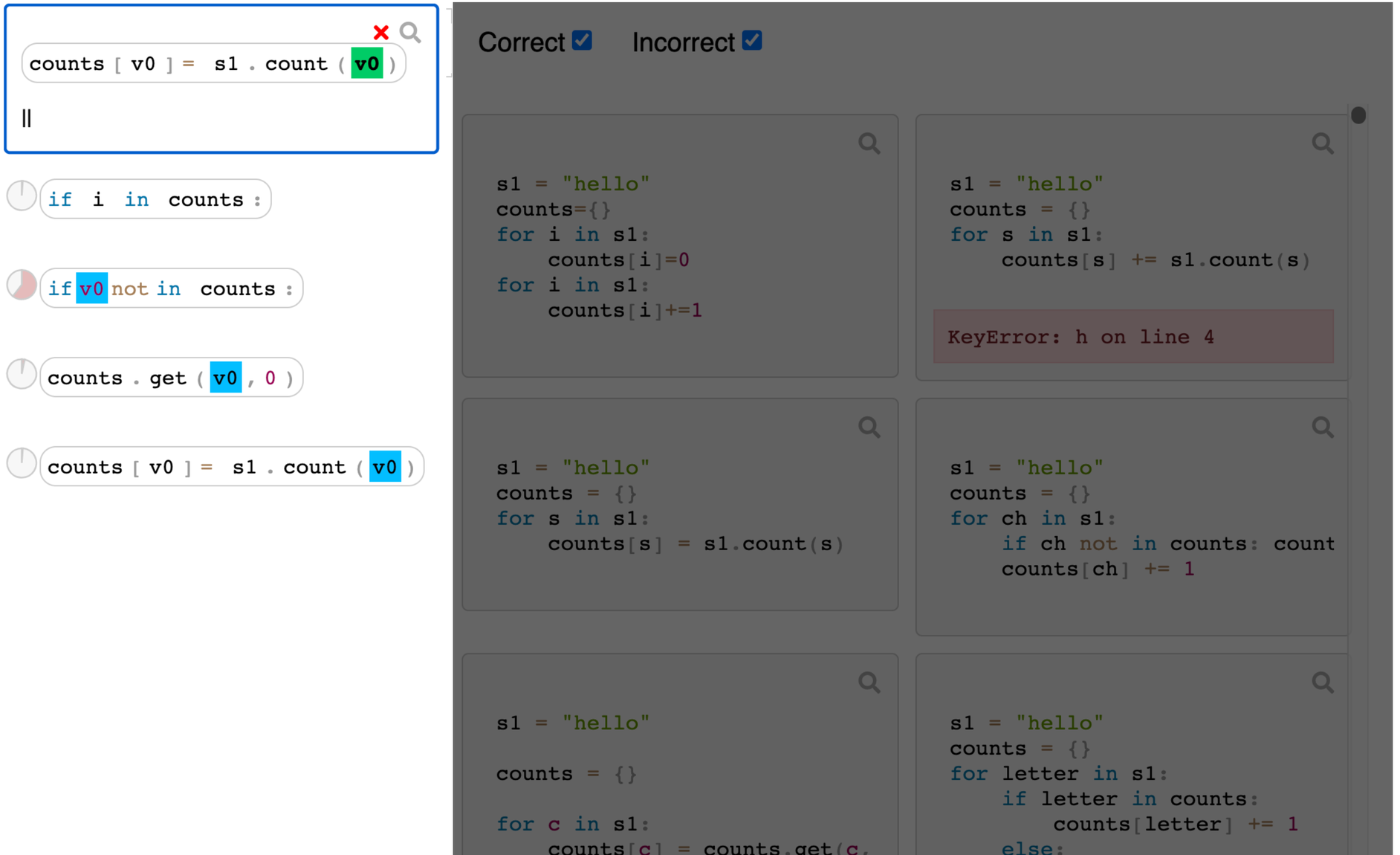
```
if v0 not in  counts :
```

```
counts . get ( v0 , 0 )
```

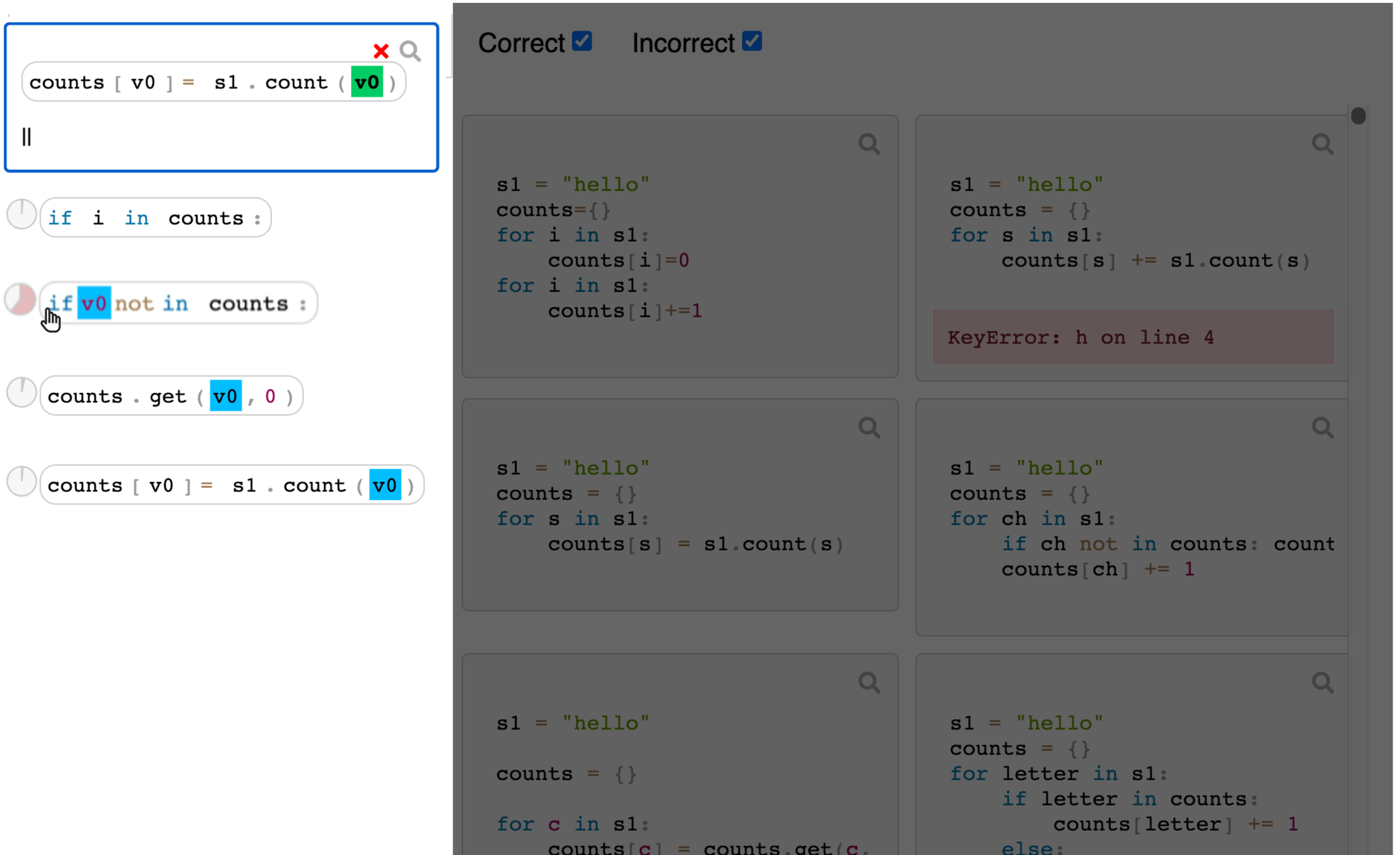
```
counts [ v0 ] =  s1 . count ( v0 )
```



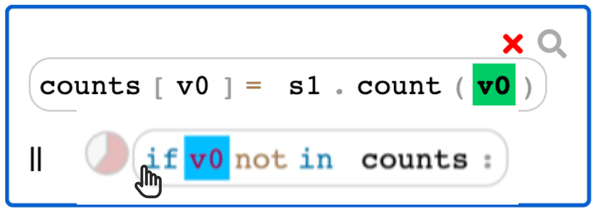
Set Maths on Queries



Set Maths on Queries



Set Maths on Queries




```
if i in counts :
```

```
if v0 not in counts :
```

```
counts . get ( v0 , 0 )
```

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

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
```

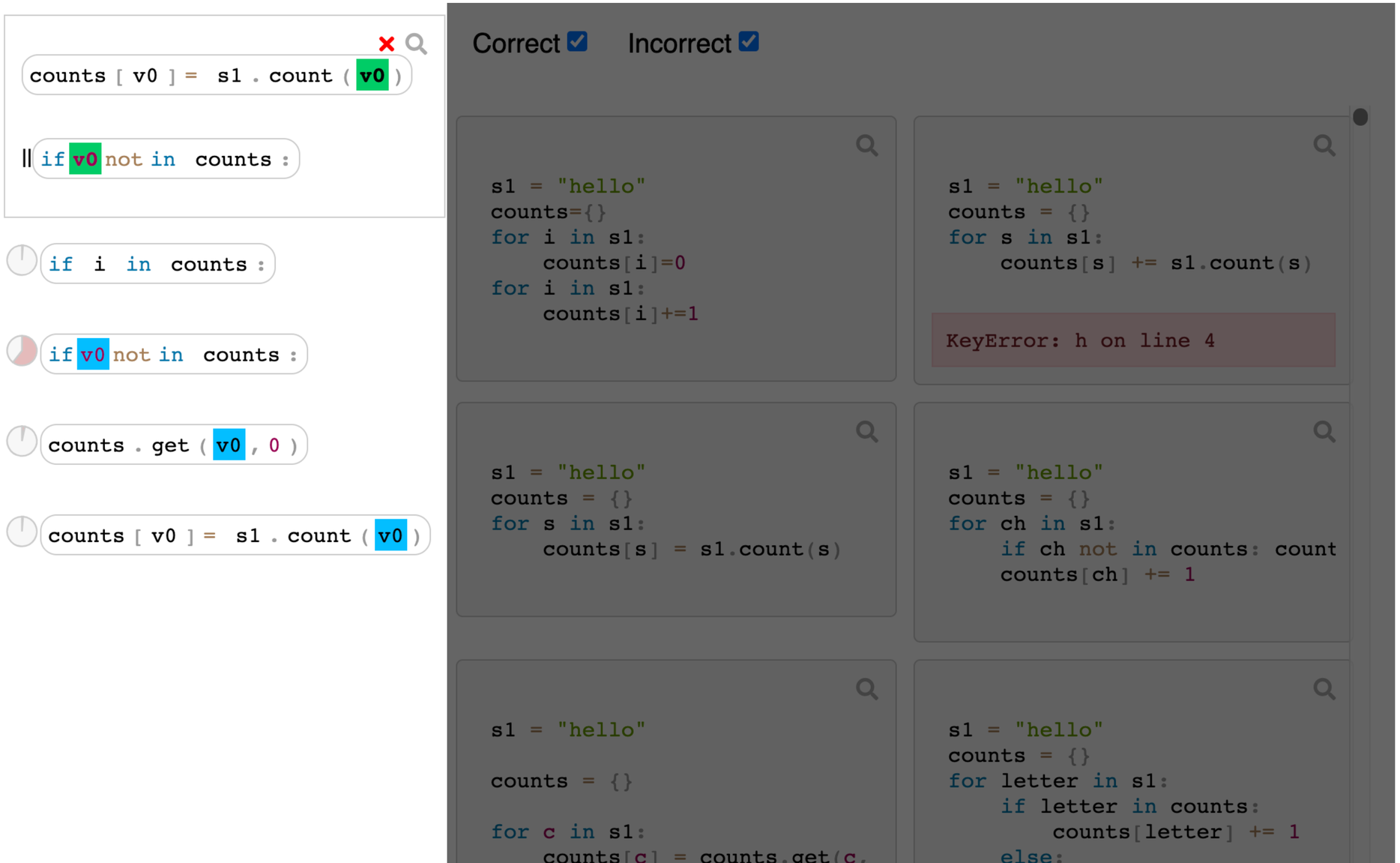
```
s1 = "hello"
counts = {}
for s in s1:
    counts[s] = s1.count(s)
```

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

```
s1 = "hello"

counts = {}

for c in s1:
```

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

Set Maths on Queries



Set Maths on Queries

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
```

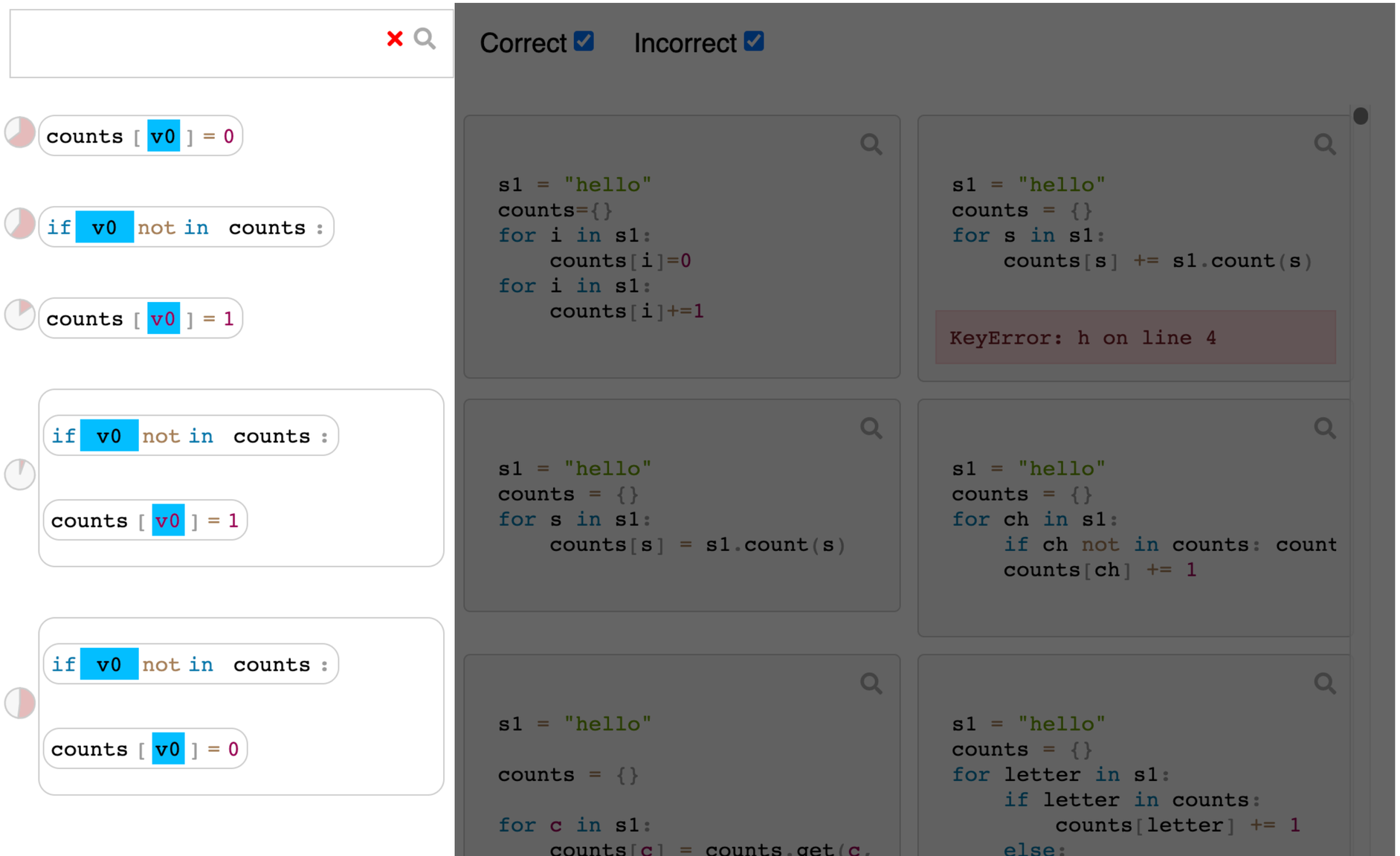
```
s1 = "hello"
counts = {}
for s in s1:
    counts[s] = s1.count(s)
```

```
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:
```

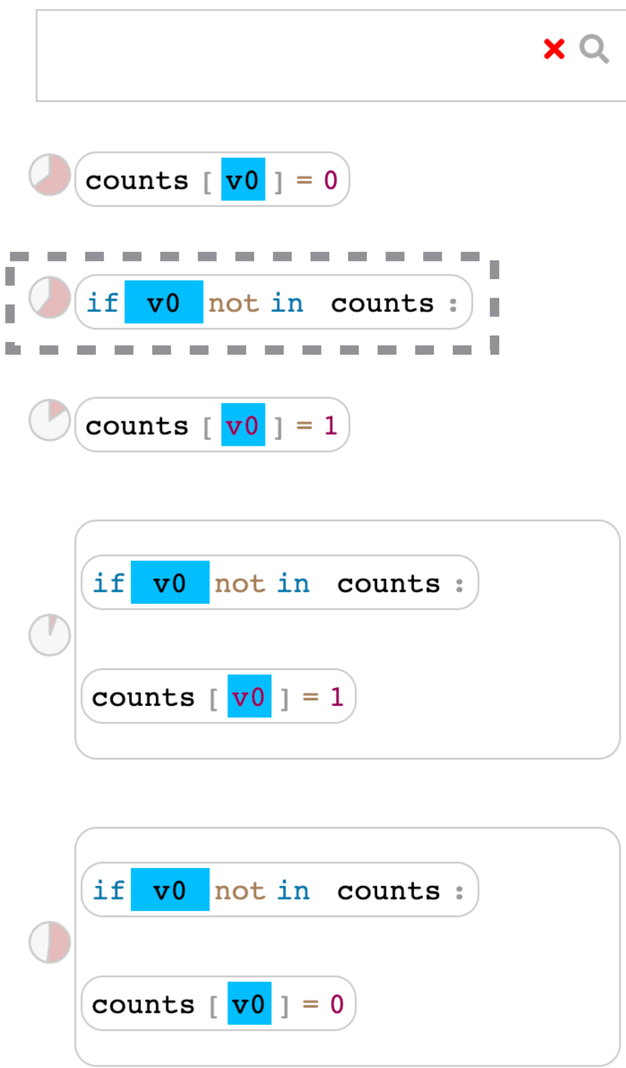
```
s1 = "hello"
counts={}
for i in s1:
    counts[i]=s1.count(i)
```

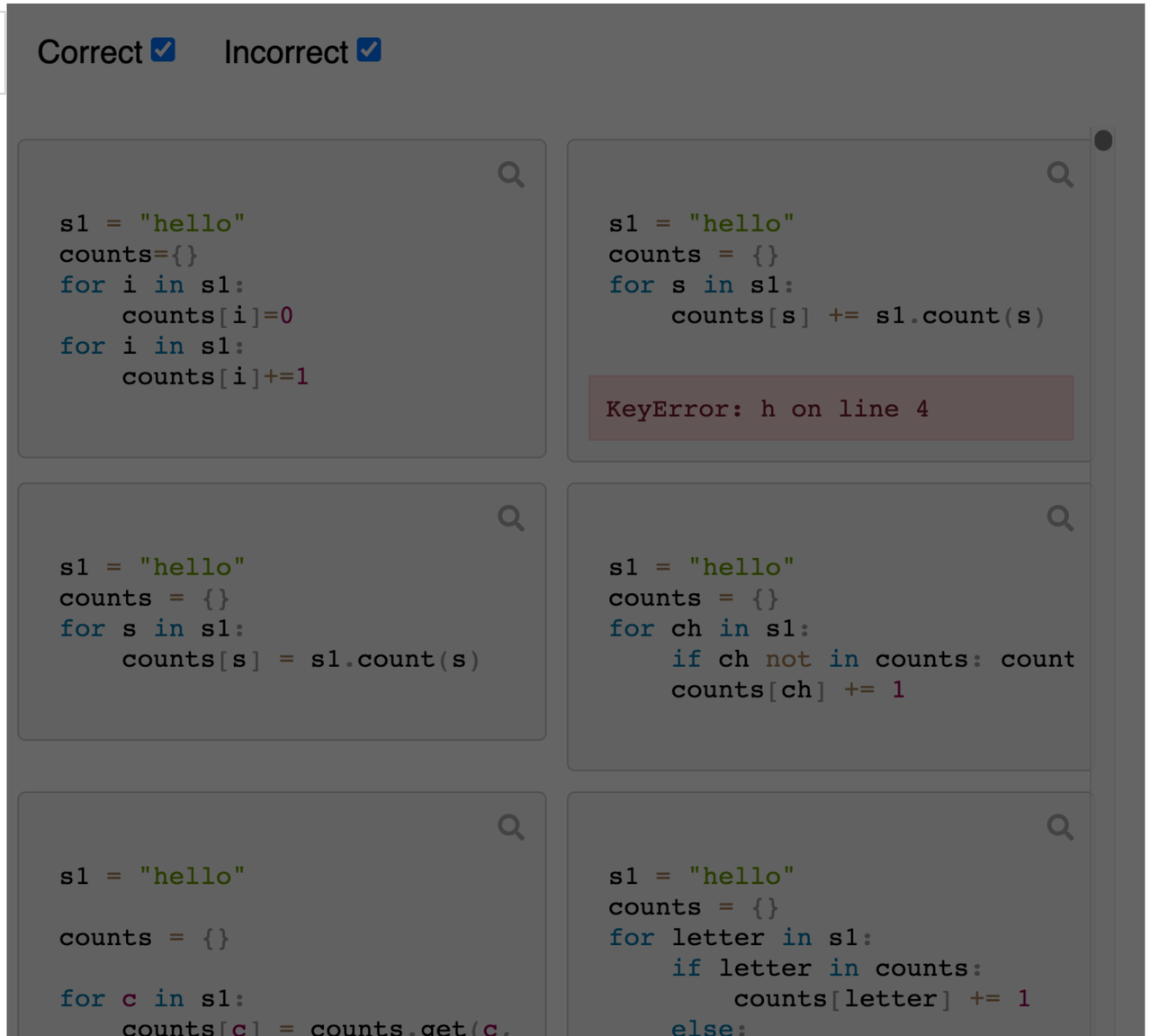
Chain Queries



Chain Queries

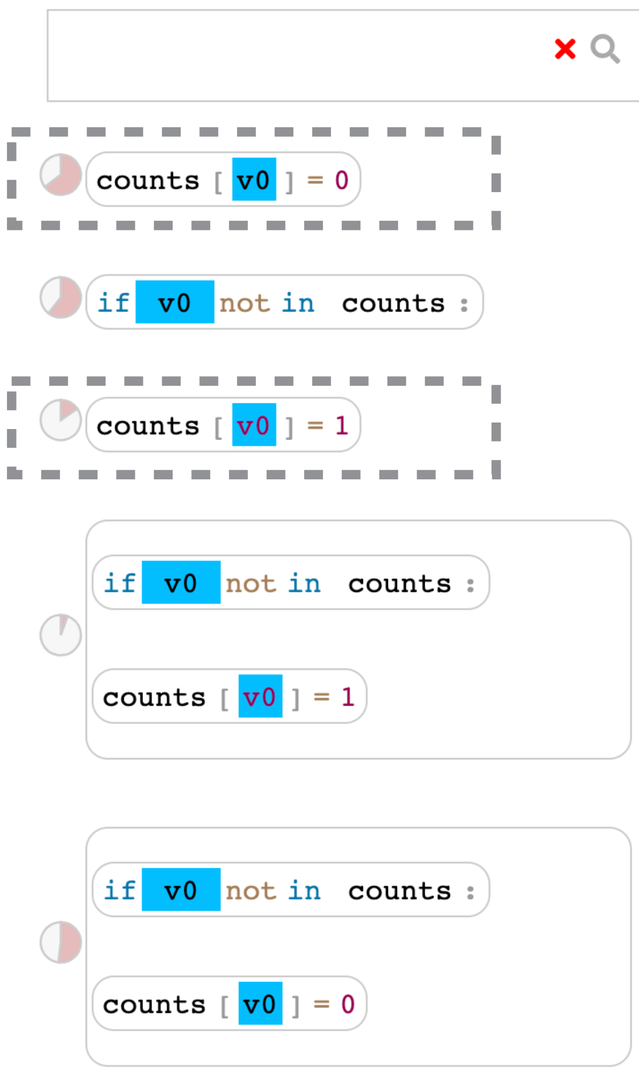
Conditional statement

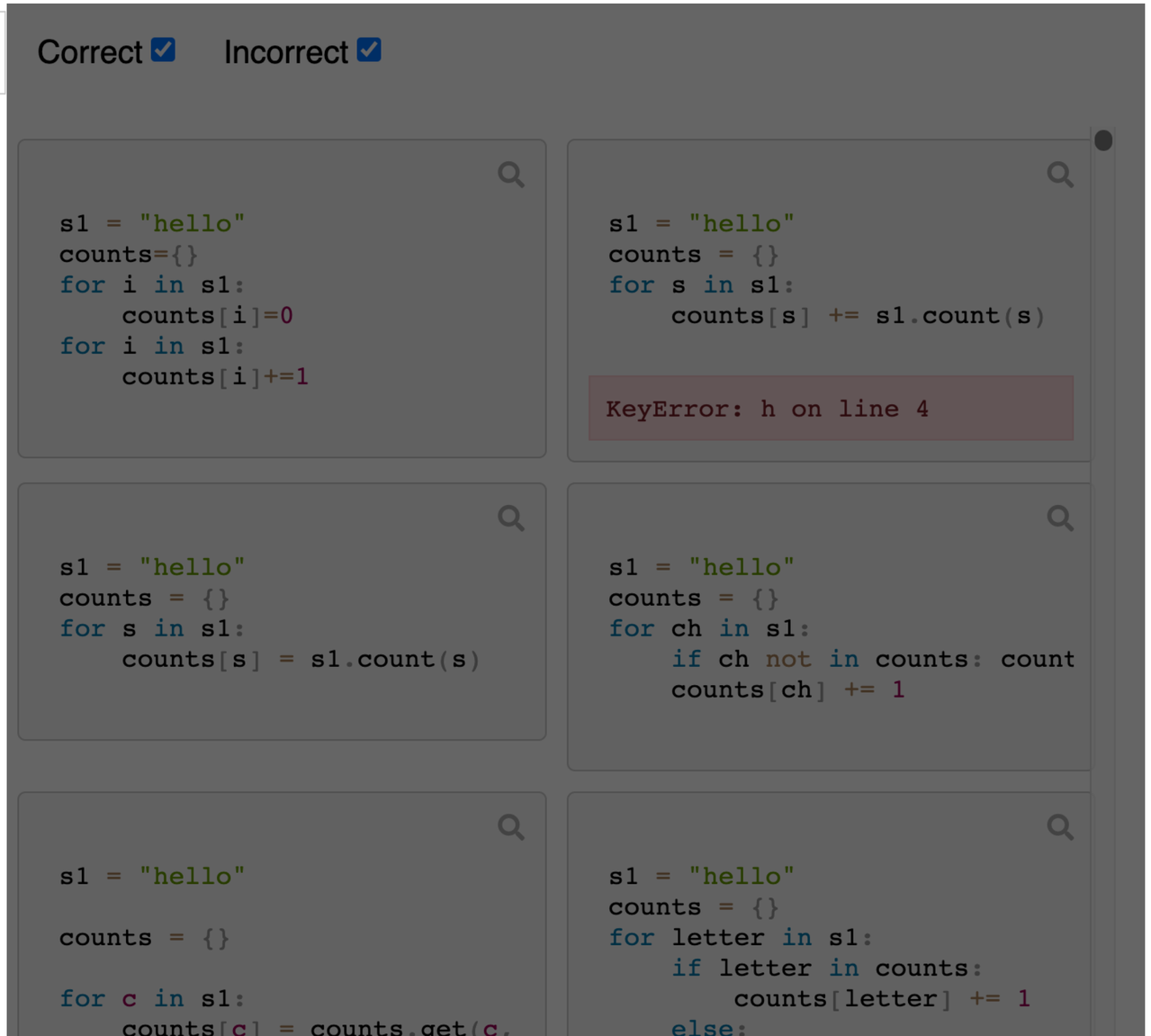




Chain Queries

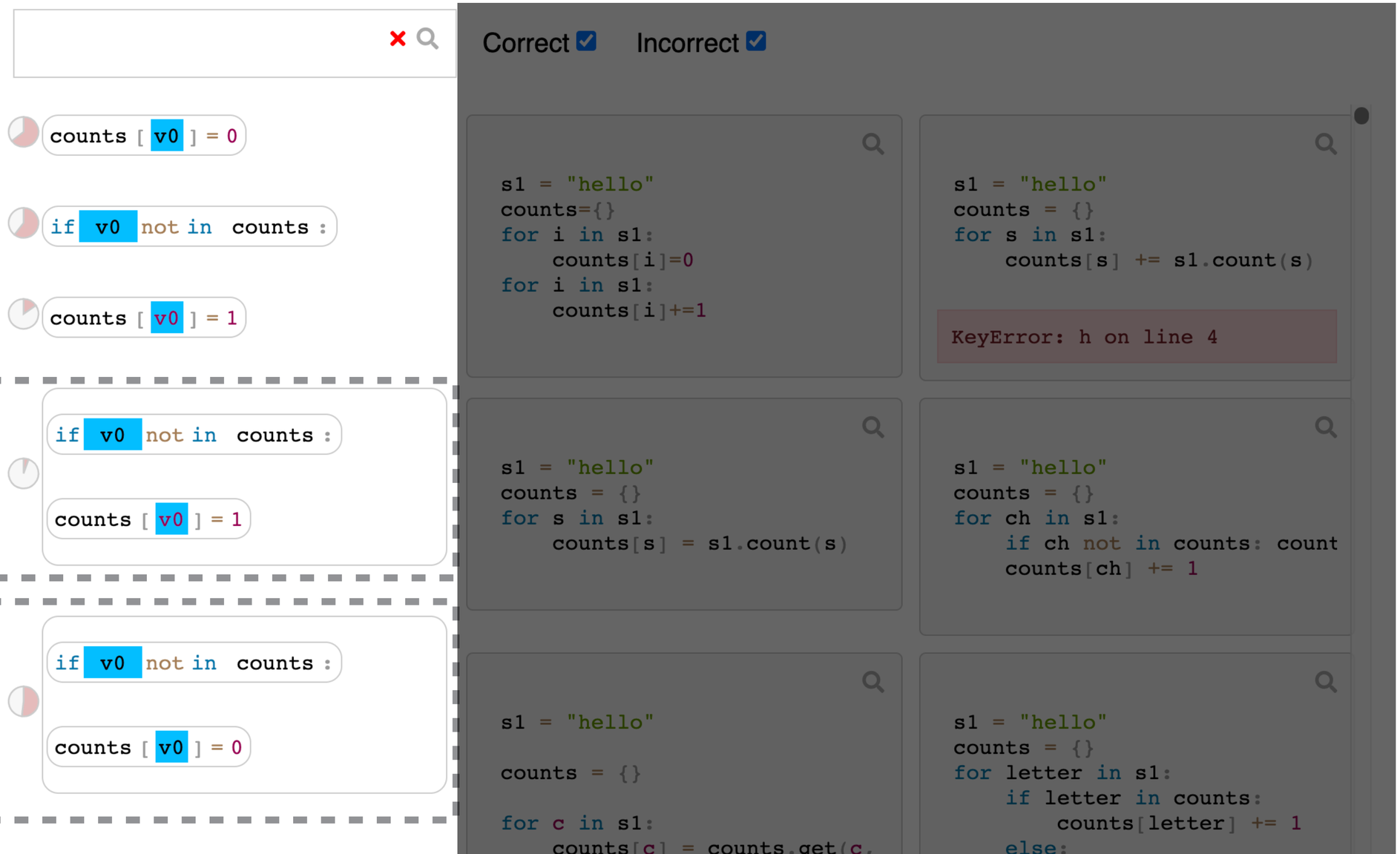
Two ways of initializing values in dict





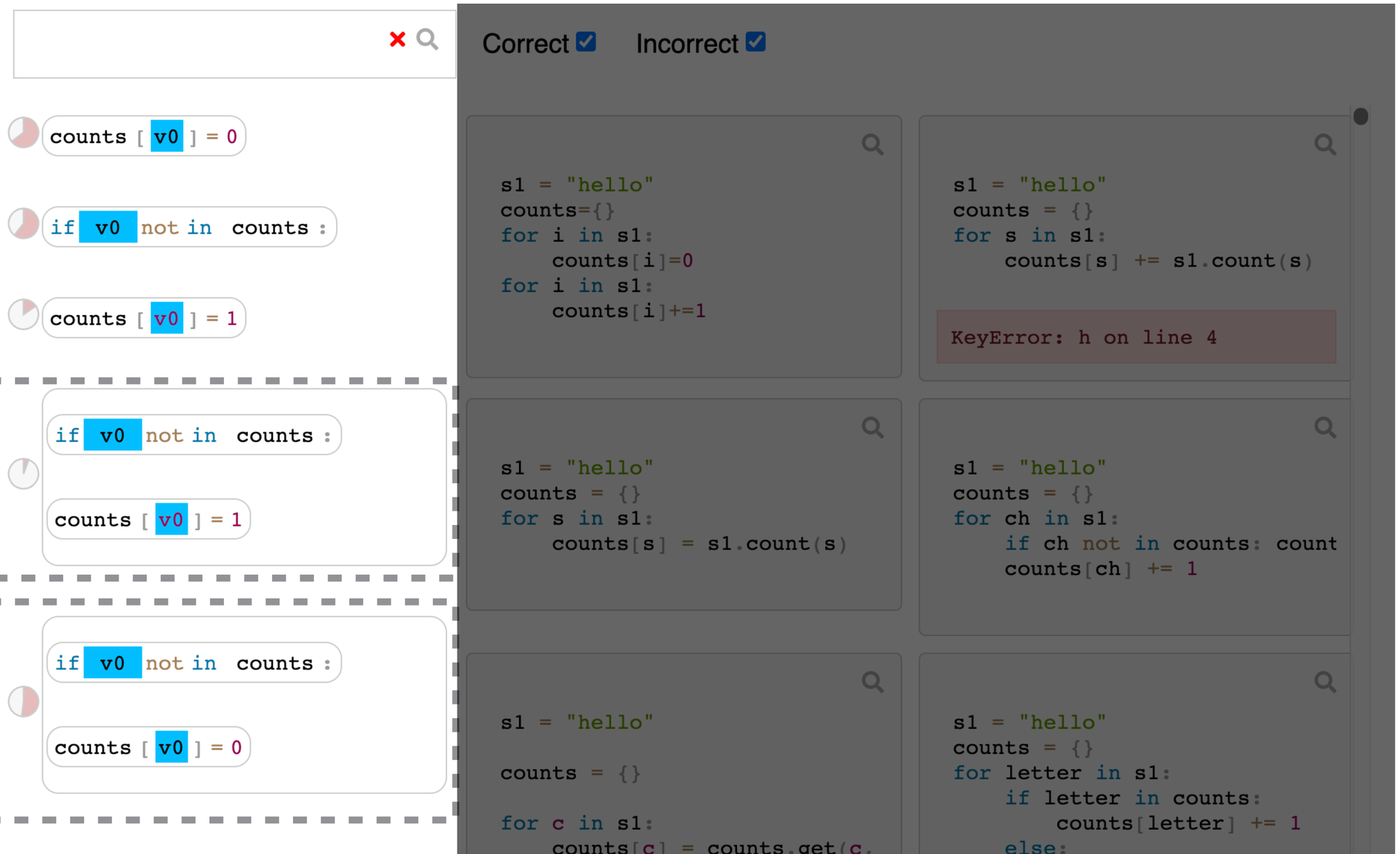
Chain Queries

Two different combinations by chaining queries together

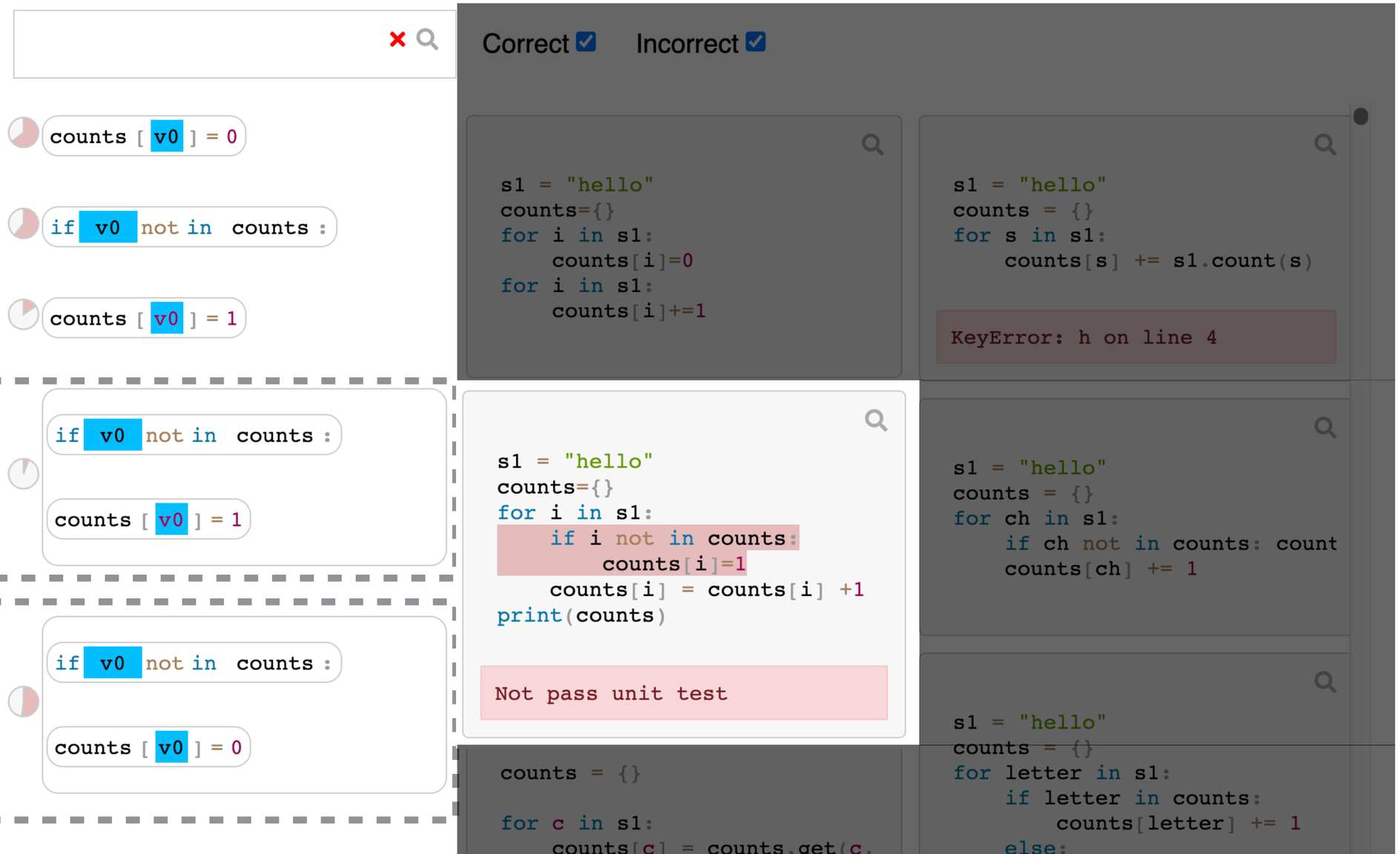


Chain Queries

Two different combinations by chaining queries together



Chain Queries



Two different combinations by chaining queries together

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 0.320	0.193 0.203	<0.0001
RunEx Baseline 2	0.892 0.616	0.193 0.141	<0.001
Baseline 1 Baseline 2	0.320 0.616	0.203 0.141	<0.001

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 0.320	0.193 0.203	<0.0001
RunEx Baseline 2	0.892 0.616	0.193 0.141	<0.001
Baseline 1 Baseline 2	0.320 0.616	0.203 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 Baseline 2	0.320 0.616	0.203 0.141	<0.001

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

Iterate and validate

Quickly create 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

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

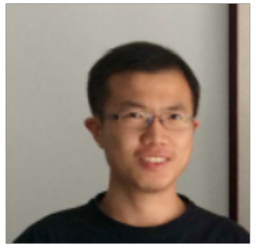
RunEx does not give users fine-grained control over how runtime values are searched - in every loop or in any loop?

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

RunEx: Augmenting Regular-Expression Code Search with **Run**time 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!