Module 01 – Exercise Class

Data Structure

Nguyen Quoc Thai

Objectives

Python Basic

- Variable
- Operators
- Condition
- Function
- Built-in Function
- For/While Loop
- If-else

Data Structure

- List
- Dictionary
- Tuple

Algorithm

- Getting Max Over Kernel
- Character Counting
- Word Counting
- Levenshtein Distance

Outline

SECTION 1

Getting Max Over Kernel

SECTION 2

Character Counting

SECTION 3

Word Counting (File)

SECTION 4

Levenshtein Distance

Getting Max Over Kernel

Description

Problem 01: Cho một list các số nguyên ***num_list*** và một sliding window (các bạn có thể tạm hiểu sliding window như là một list có kích thước nhỏ hơn ***num_list***) có kích thước size ***k*** di chuyển từ trái sang phải. Mỗi lần dịch chuyển 1 vị trí sang phải có thể nhìn thấy được ***k*** số trong ***num_list*** và tìm số lớn nhất trong ***k*** số này sau mỗi lần trượt. ***k*** phải lớn hơn hoặc bằng 1. Các bạn hãy viết chương trình Python giải quyết vấn đề trên.

Example:

- **Input:**

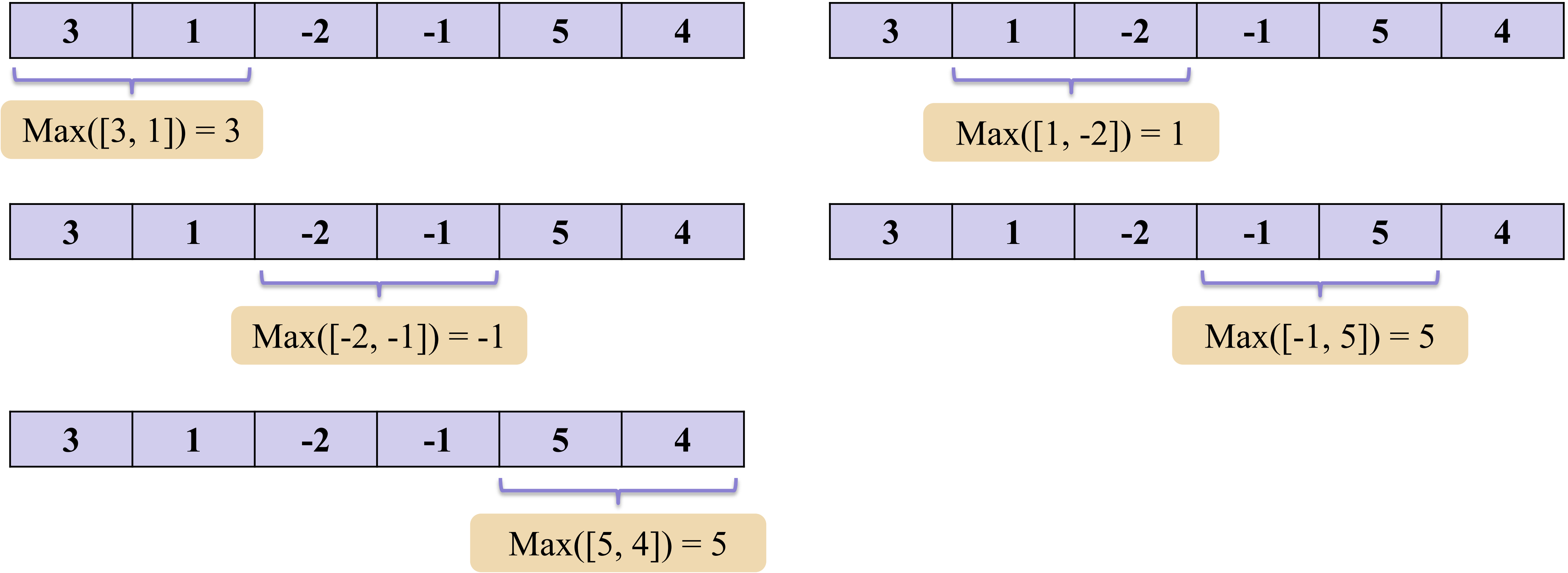
  num_list = [3, 4, 5, 1, −44, 5, 10, 12, 33, 1]
  k = 3
- **Output:** [5, 5, 5, 5, 10, 12, 33, 33]

Getting Max Over Kernel

Solution

Use *slicing* and *max()* function

k = 2



3	1	-2	-1	5	4

Max([3, 1]) = 3

Max([1, -2]) = 1

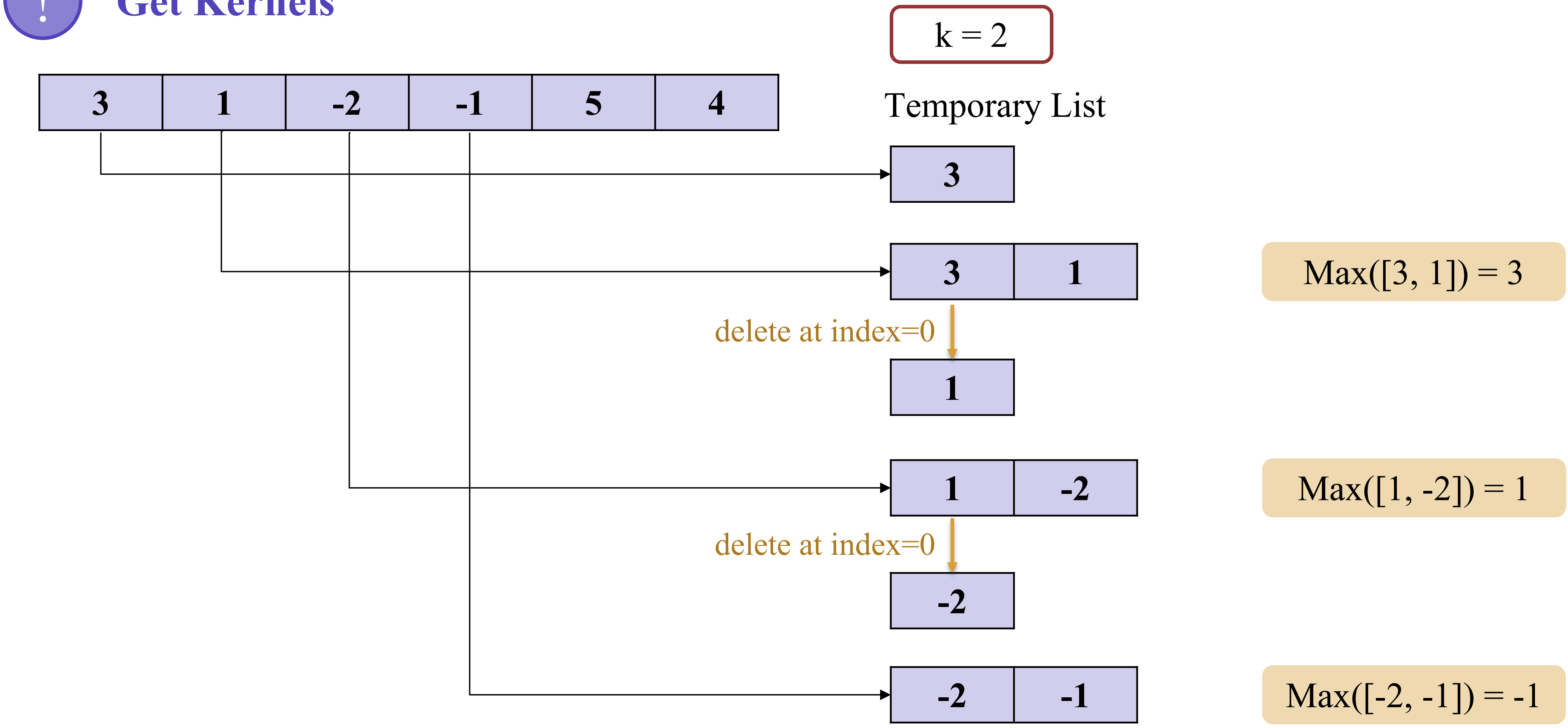
Max([-2, -1]) = -1

Max([-1, 5]) = 5

Max([5, 4]) = 5

Getting Max Over Kernel

Get Kernels



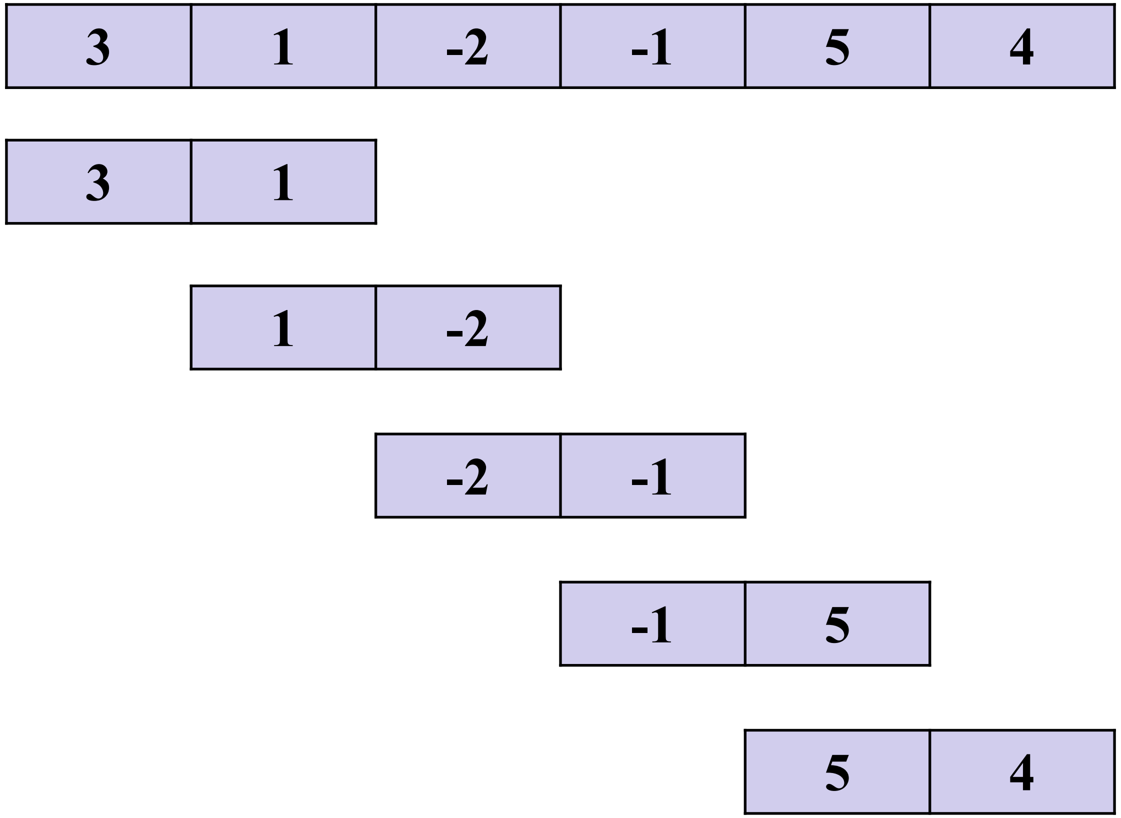
Max([3, 1]) = 3

Max([1, -2]) = 1

Max([-2, -1]) = -1

Getting Max Over Kernel

Get Kernels



3 1 -2 -1 5 4

3 1

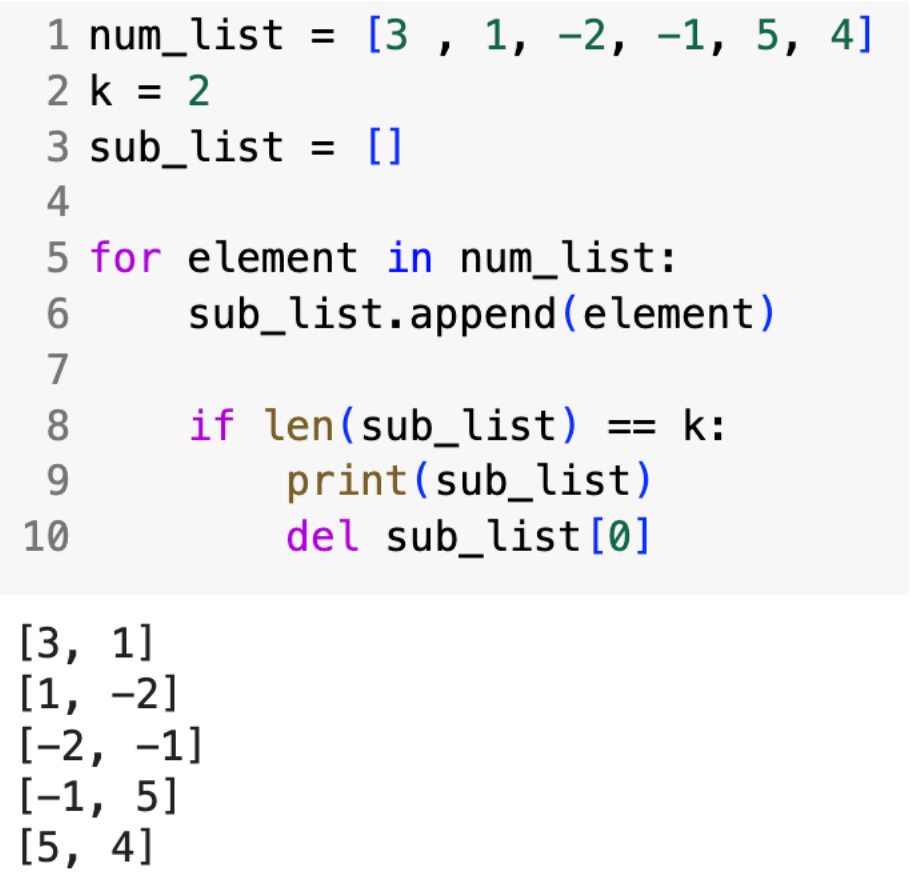
1 -2

-2 -1

-1 5

5 4

k = 2

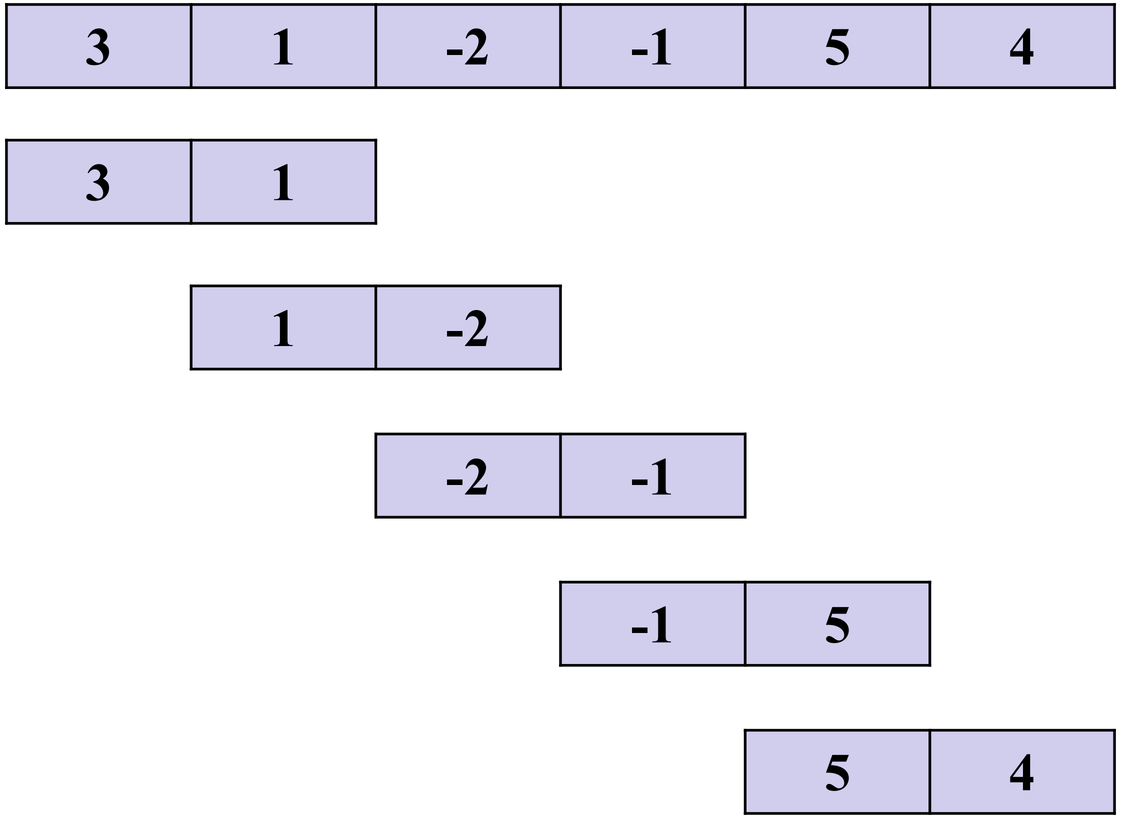


```python
num_list = [3 , 1, -2, -1, 5, 4]
k = 2
sub_list = []

for element in num_list:
    sub_list.append(element)

    if len(sub_list) == k:
        print(sub_list)
        del sub_list[0]
```

```
[3, 1]
[1, -2]
[-2, -1]
[-1, 5]
[5, 4]
```

Getting Max Over Kernel

Get Kernels – Solution #1



3 | 1 | -2 | -1 | 5 | 4

3 | 1

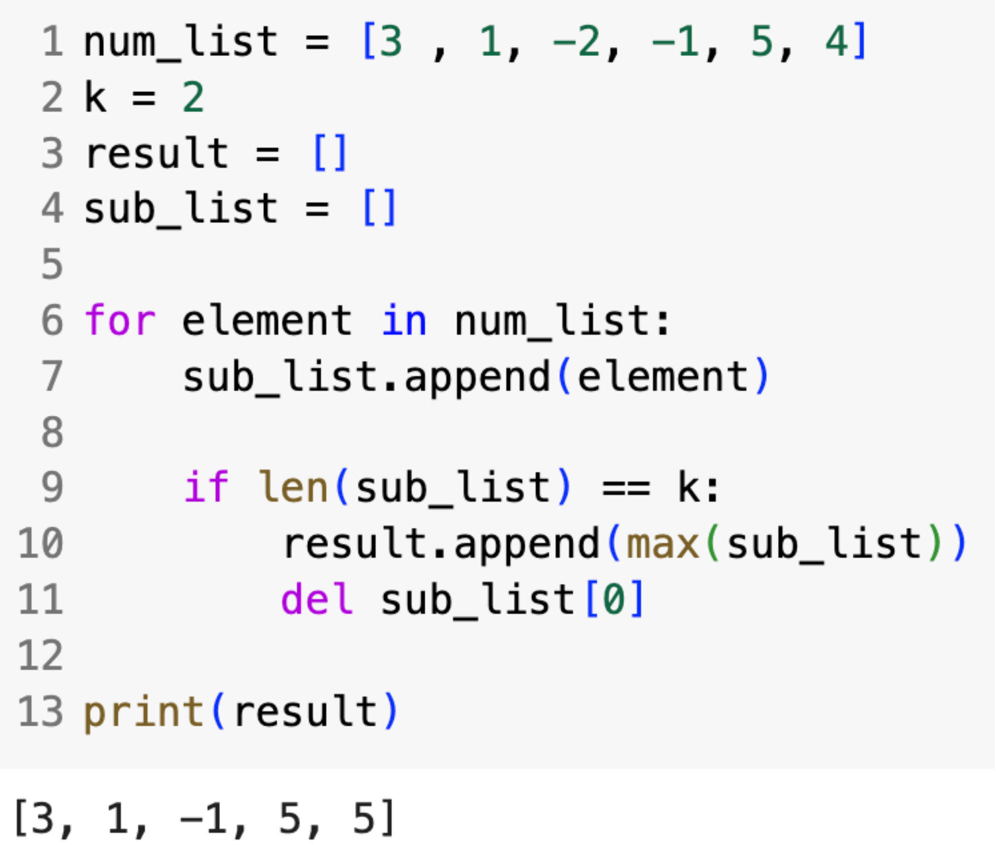
1 | -2

-2 | -1

-1 | 5

5 | 4

k = 2



```
num_list = [3 , 1, -2, -1, 5, 4]
k = 2
result = []
sub_list = []

for element in num_list:
    sub_list.append(element)

    if len(sub_list) == k:
        result.append(max(sub_list))
        del sub_list[0]

print(result)
```

```
[3, 1, -1, 5, 5]
```

Getting Max Over Kernel

! Slicing – Solution #2

3	1	-2	-1	5	4

k = 2

list[start:end]

list[0:2]

3	1

list[1:3]

1	-2

list[2:4]

-2	-1

list[3:5]

-1	5

list[4:6]

5	4

Start Index	0	1	2	3	4
End Index	2	3	4	5	6

0 => len(list) - k

k => len(list)

Getting Max Over Kernel

Slicing – Solution #2

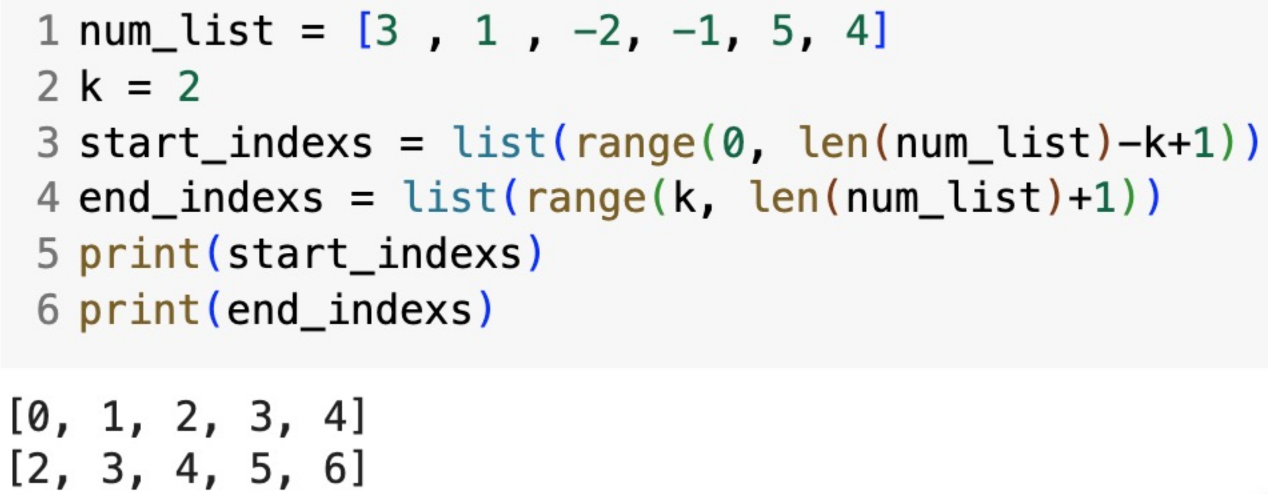
3	1	-2	-1	5	4

k = 2

Start Index	0	1	2	3	4
End Index	2	3	4	5	6

0 => len(list) - k

k => len(list)



```
num_list = [3 , 1 , -2, -1, 5, 4]
k = 2
start_indexs = list(range(0, len(num_list)-k+1))
end_indexs = list(range(k, len(num_list)+1))
print(start_indexs)
print(end_indexs)
```

```
[0, 1, 2, 3, 4]
[2, 3, 4, 5, 6]
```

Getting Max Over Kernel

Slicing – Solution #2

3	1	-2	-1	5	4

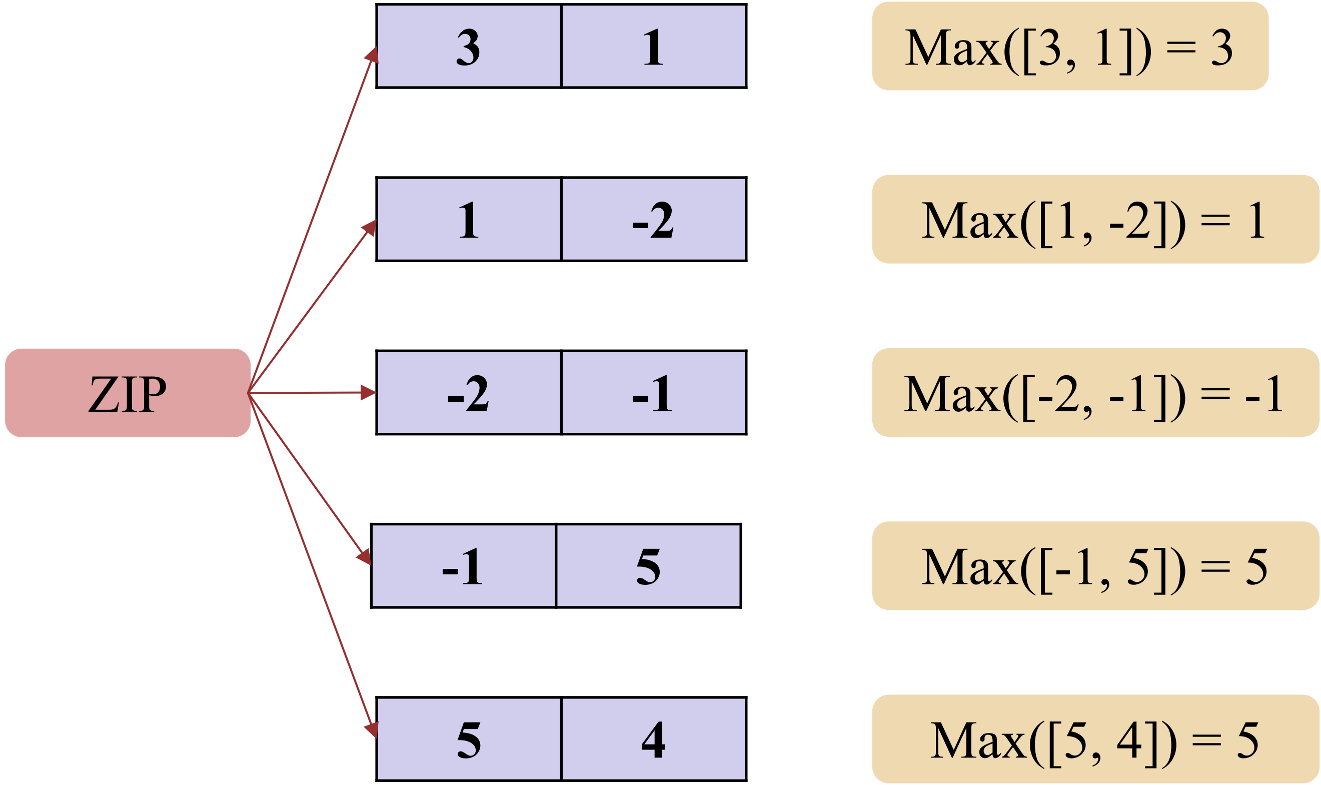
k = 2

Start Index

0	1	2	3	4

End Index

2	3	4	5	6



ZIP

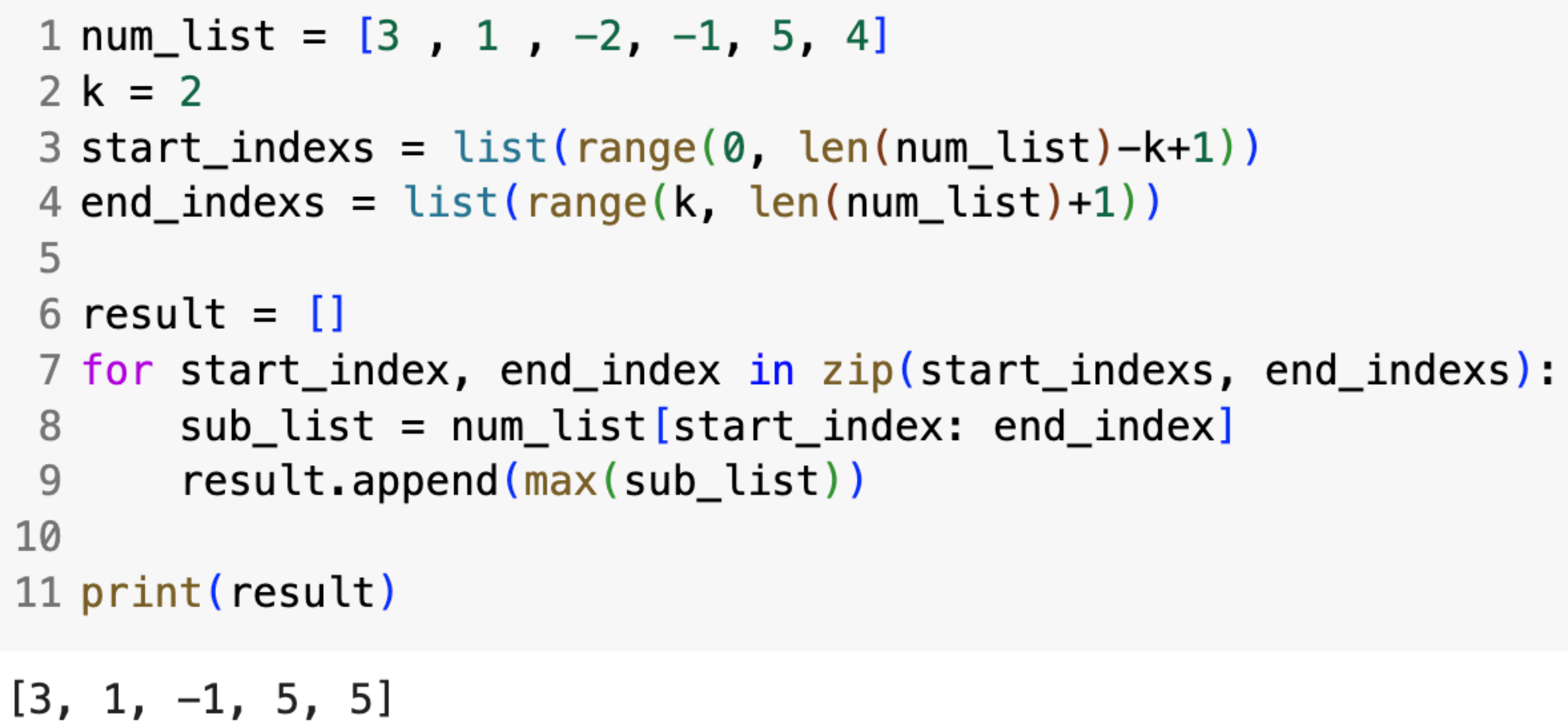
3	1	Max([3, 1]) = 3
1	-2	Max([1, -2]) = 1
-2	-1	Max([-2, -1]) = -1
-1	5	Max([-1, 5]) = 5
5	4	Max([5, 4]) = 5

Getting Max Over Kernel

Slicing

3	**1**	**-2**	**-1**	**5**	**4**

k = 2



```python
num_list = [3 , 1 , -2, -1, 5, 4]
k = 2
start_indexs = list(range(0, len(num_list)-k+1))
end_indexs = list(range(k, len(num_list)+1))

result = []
for start_index, end_index in zip(start_indexs, end_indexs):
    sub_list = num_list[start_index: end_index]
    result.append(max(sub_list))

print(result)
```

```
[3, 1, -1, 5, 5]
```

Outline

SECTION 1
Getting Max Over Kernel

SECTION 2
Character Counting

SECTION 3
Word Counting (File)

SECTION 4
Levenshtein Distance

Character Counting

Description

Problem 01: Viết thuật toán trả về một dictionary đếm số lượng chữ xuất hiện trong một từ, với key là chữ cái và value là số lần xuất hiện.

Input: một từ

Output: dictionary đếm số lần các chữ xuất hiện

Note: Giả sử các từ nhập vào đều có các chữ cái thuộc [a-z] hoặc [A-Z]

Example:

- **Input:**

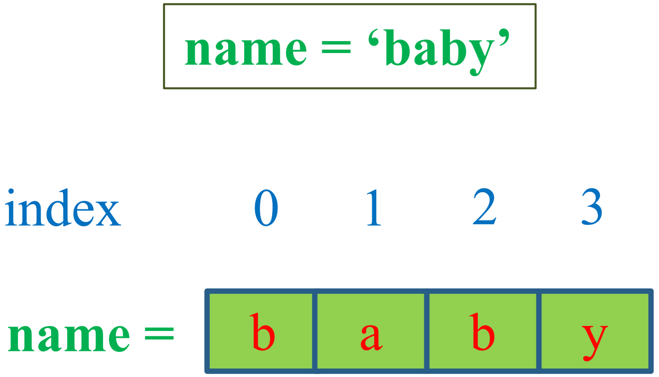
  word = 'baby'

- **Output:**

```
{'b': 2, 'a': 1, 'y': 1}
```

Character Counting

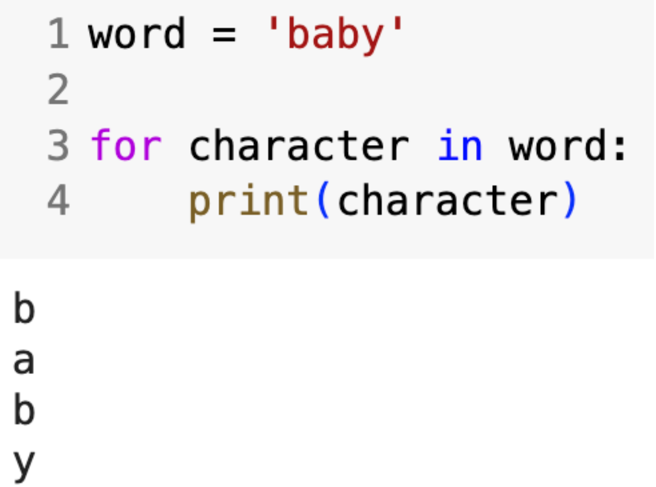
Solution



name = ‘baby’

index 0 1 2 3

name = b a b y



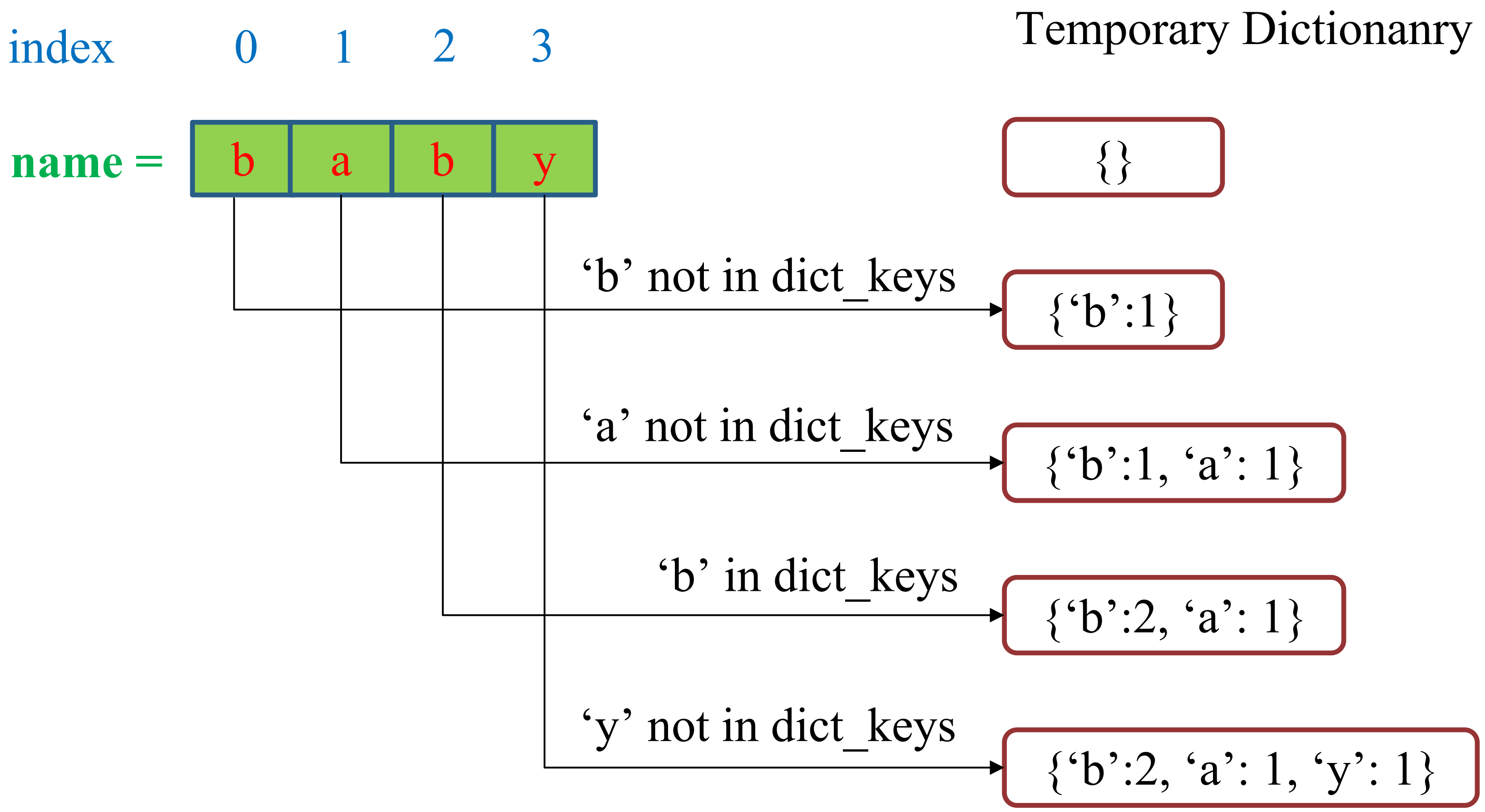
```
word = 'baby'

for character in word:
    print(character)
```

```
b
a
b
y
```

Character Counting

Solution

name = ‘baby’



Temporary Dictionanry

index 0 1 2 3

name = b a b y

{}

‘b’ not in dict_keys

{‘b’:1}

‘a’ not in dict_keys

{‘b’:1, ‘a’: 1}

‘b’ in dict_keys

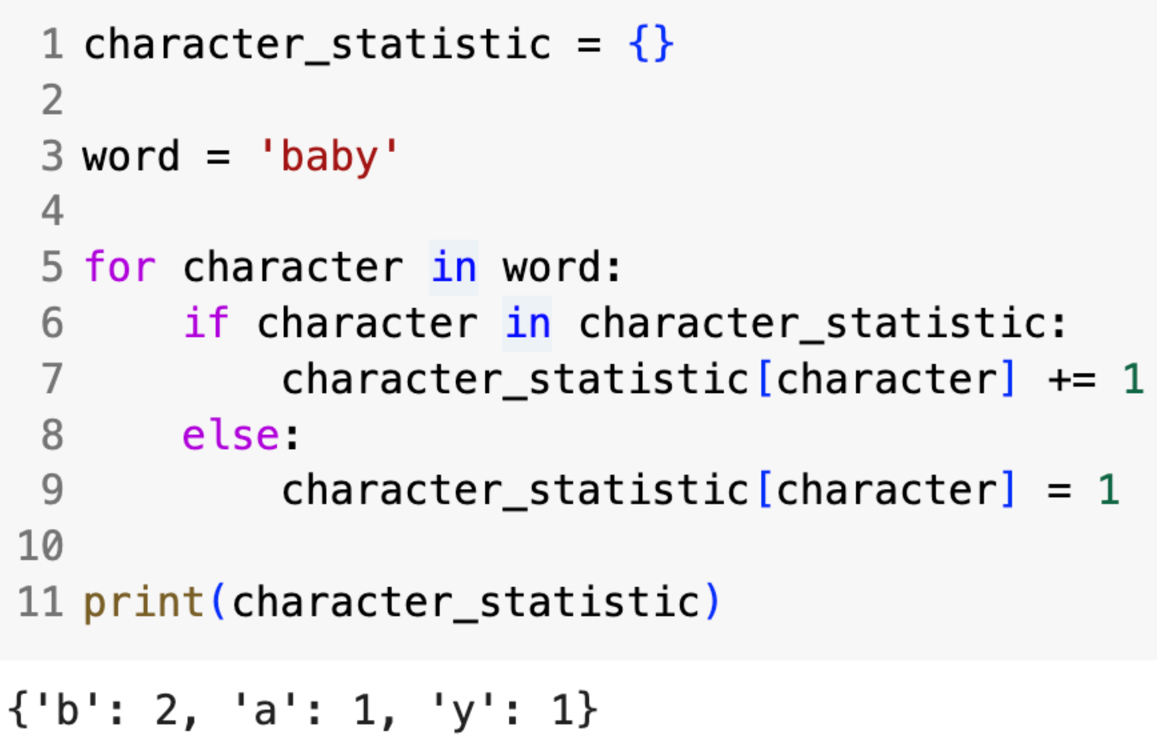
{‘b’:2, ‘a’: 1}

‘y’ not in dict_keys

{‘b’:2, ‘a’: 1, ‘y’: 1}

Character Counting

Solution



```python
character_statistic = {}

word = 'baby'

for character in word:
    if character in character_statistic:
        character_statistic[character] += 1
    else:
        character_statistic[character] = 1

print(character_statistic)
```

```
{'b': 2, 'a': 1, 'y': 1}
```

Character Counting

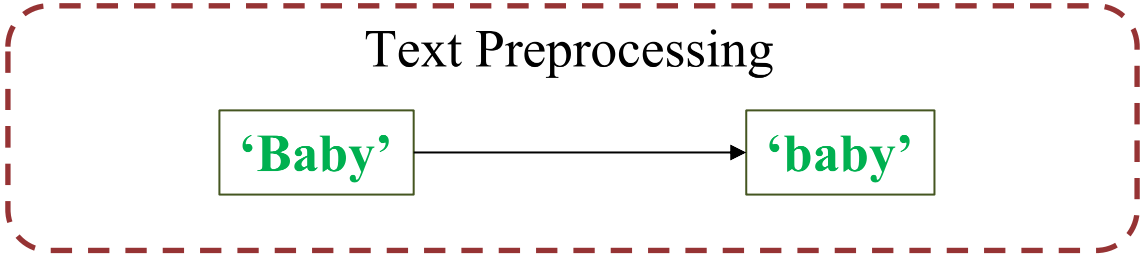
Extension

‘Baby’ `{'B': 1, 'a': 1, 'b': 1, 'y': 1}`

≠

‘baby’ `{'b': 2, 'a': 1, 'y': 1}`

‘Baby’ and ‘baby’: the same meaning in text

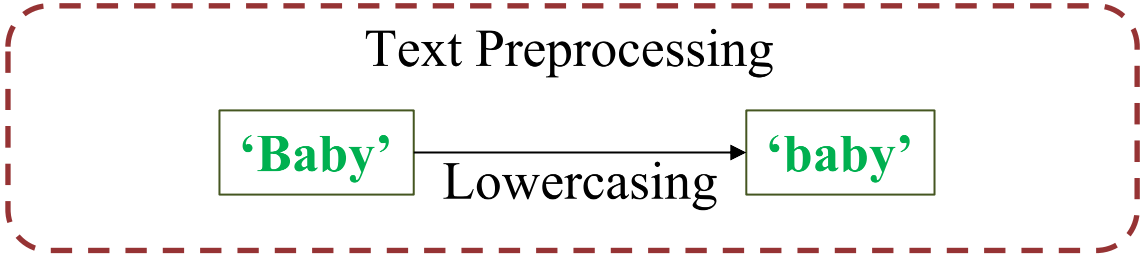


Text Preprocessing

‘Baby’ → ‘baby’

Character Counting

Extension

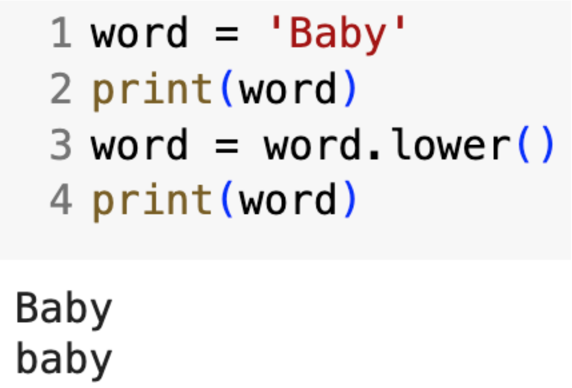


Text Preprocessing

‘Baby’

Lowercasing

‘baby’

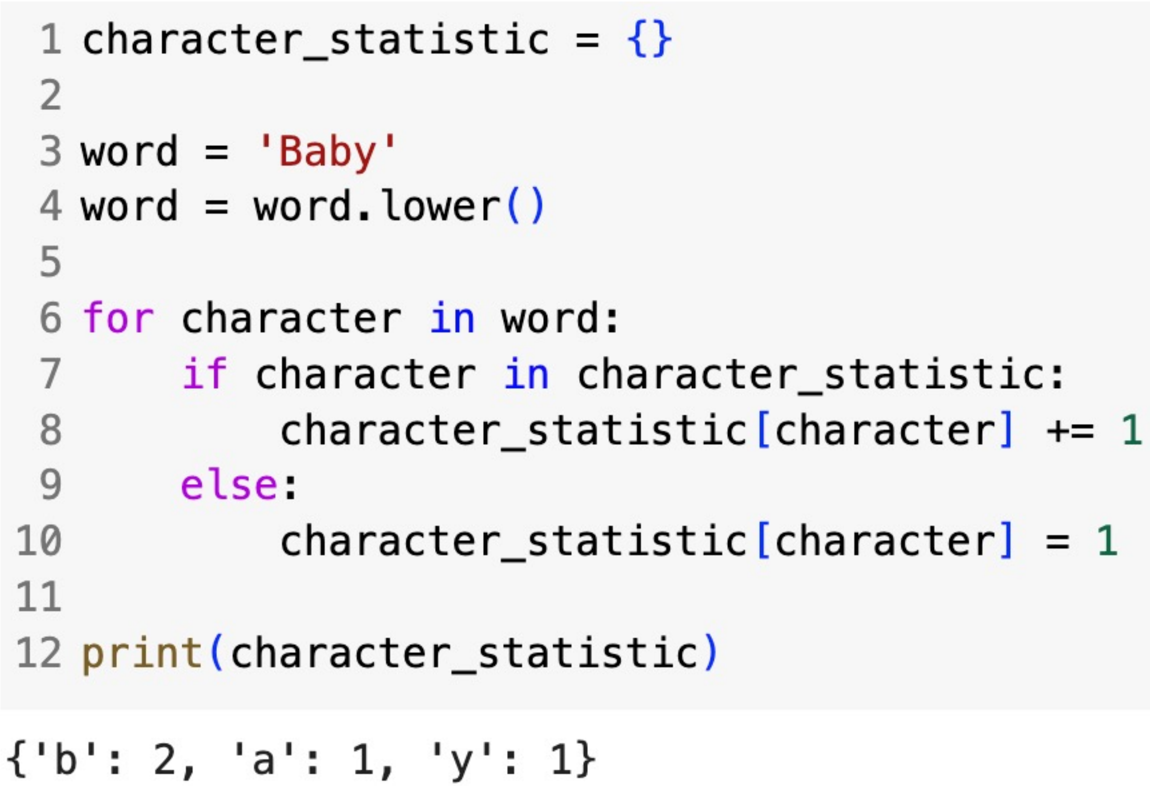


```python
word = 'Baby'
print(word)
word = word.lower()
print(word)
```

```
Baby
baby
```

Character Counting

Extension



```python
character_statistic = {}

word = 'Baby'
word = word.lower()

for character in word:
    if character in character_statistic:
        character_statistic[character] += 1
    else:
        character_statistic[character] = 1

print(character_statistic)
```

```
{'b': 2, 'a': 1, 'y': 1}
```

Outline

SECTION 1

Getting Max Over Kernel

SECTION 2

Character Counting

SECTION 3

Word Counting (File)

SECTION 4

Levenshtein Distance

Word Counting

Description

Problem: Viết thuật toán đọc các câu trong một file txt, đếm số lượng các từ xuất hiện và trả về một dictionary với key là từ và value là số lần từ đó xuất hiện.

Input: Đường dẫn đến file txt

Output: Dictionary đếm số lần các từ xuất hiện

Note:

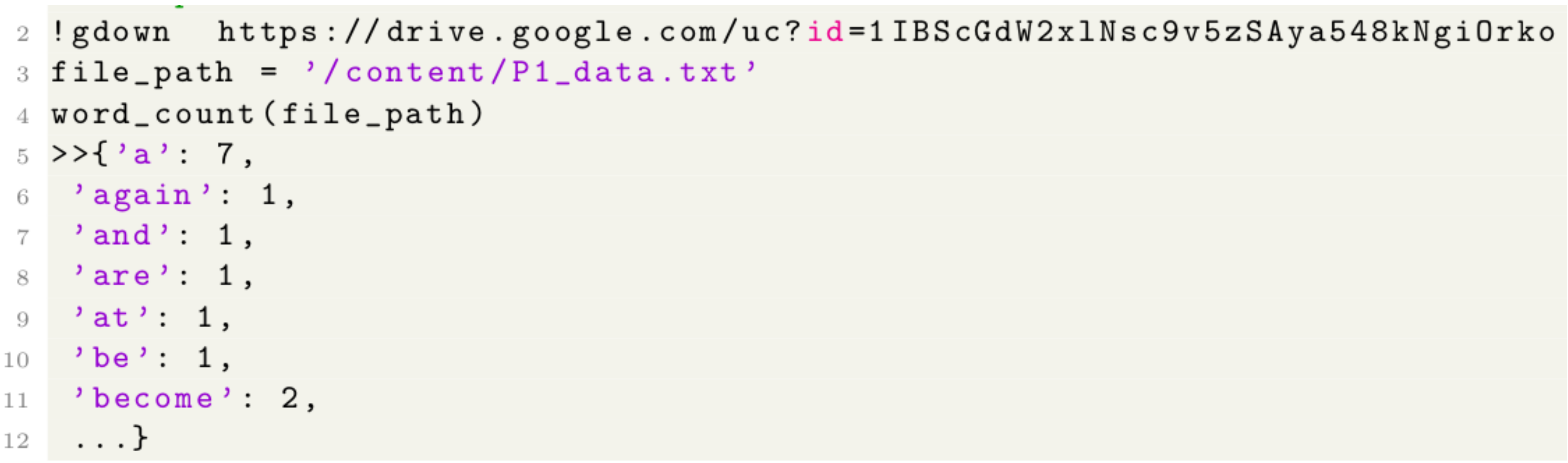
Giả sử các từ trong file txt đều có các chữ cái thuộc [a-z] hoặc [A-Z]

Không cần các thao tác xử lý string phức tạp nhưng cần xử lý các từ đều là viết thường

File: https://drive.google.com/uc?id=1IBScGdW2xlNsc9v5zSAya548kNgiOrko

Word Counting

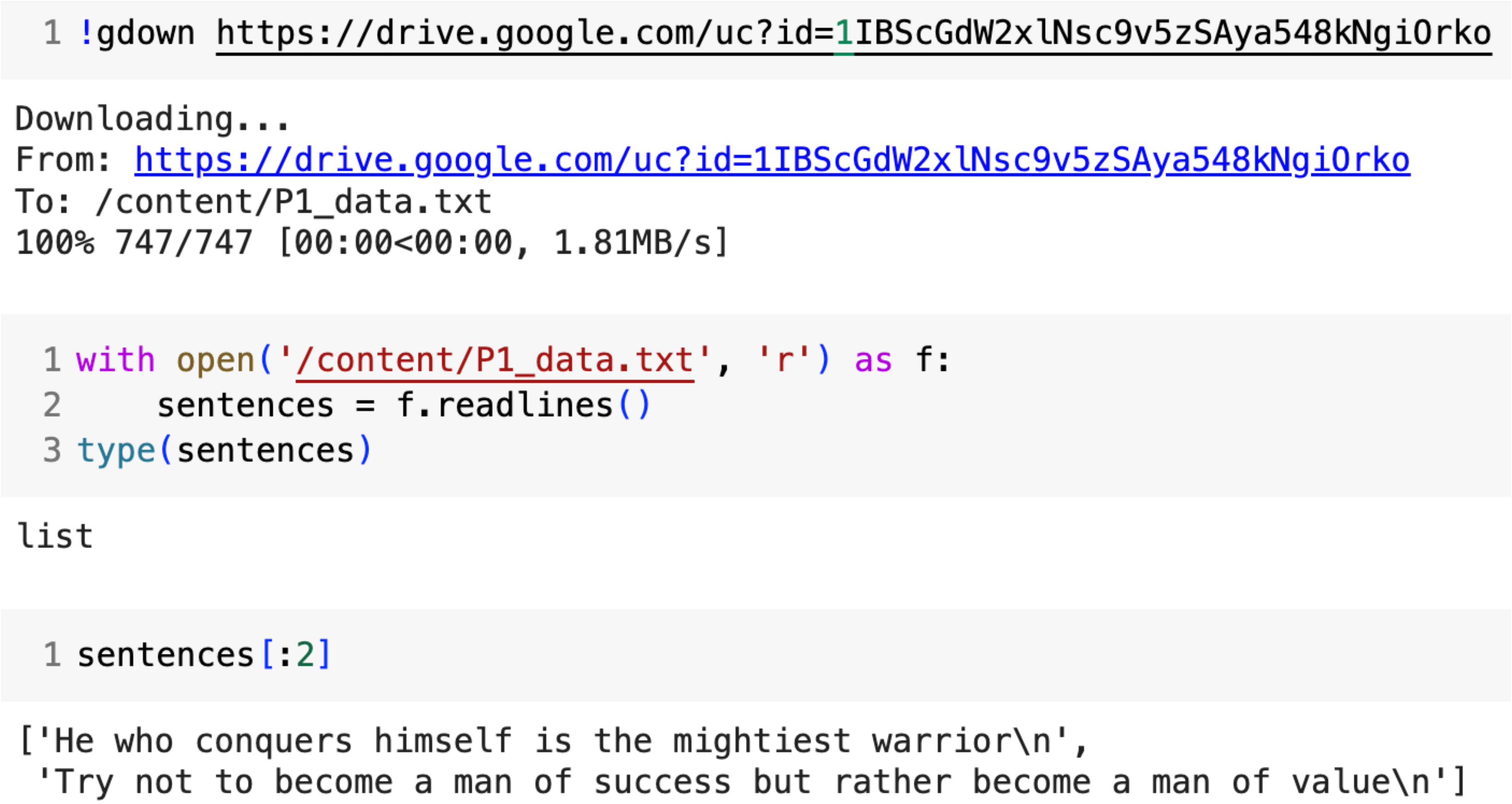
Example



```
!gdown  https://drive.google.com/uc?id=1IBScGdW2xlNsc9v5zSAya548kNgi0rko
file_path = ’/content/P1_data.txt’
word_count(file_path)
>>{’a’: 7,
 ’again’: 1,
 ’and’: 1,
 ’are’: 1,
 ’at’: 1,
 ’be’: 1,
 ’become’: 2,
 ...}
```

Word Counting

Read File



```
!gdown https://drive.google.com/uc?id=1IBScGdW2xlNsc9v5zSAya548kNgiOrko
```

```
Downloading...
From: https://drive.google.com/uc?id=1IBScGdW2xlNsc9v5zSAya548kNgiOrko
To: /content/P1_data.txt
100% 747/747 [00:00<00:00, 1.81MB/s]
```

```
with open('/content/P1_data.txt', 'r') as f:
    sentences = f.readlines()
type(sentences)
```

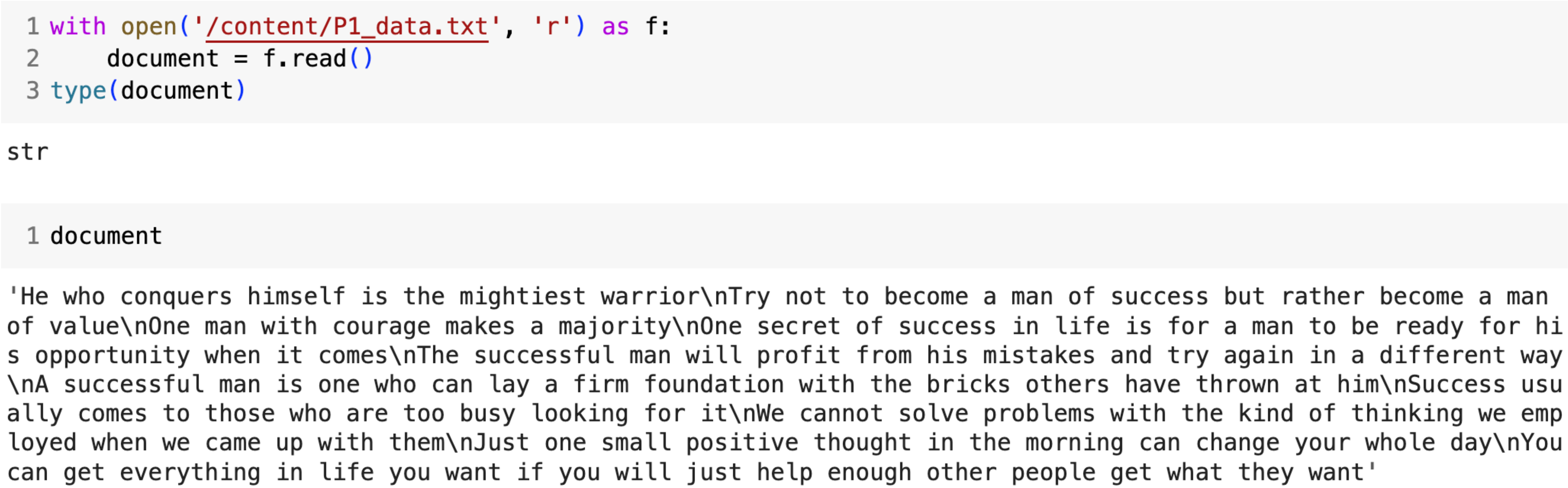
```
list
```

```
sentences[:2]
```

```
['He who conquers himself is the mightiest warrior\n',
 'Try not to become a man of success but rather become a man of value\n']
```

Word Counting

Read File



```
with open('/content/P1_data.txt', 'r') as f:
    document = f.read()
type(document)
```

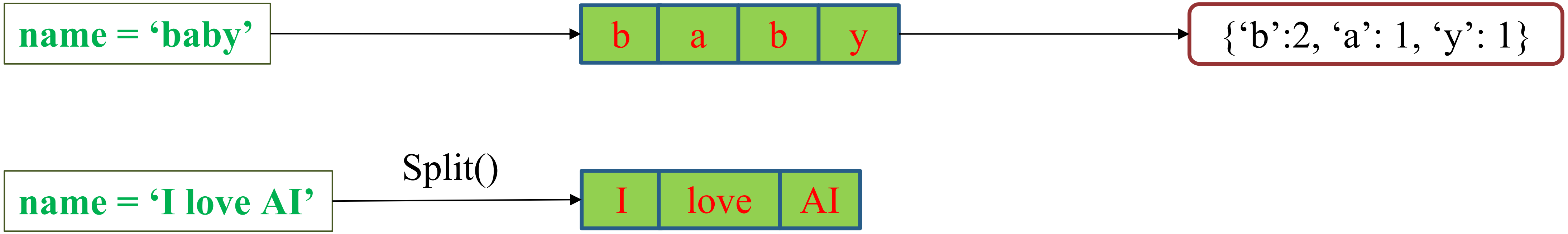
```
str
```

```
document
```

```
'He who conquers himself is the mightiest warrior\nTry not to become a man of success but rather become a man
of value\nOne man with courage makes a majority\nOne secret of success in life is for a man to be ready for hi
s opportunity when it comes\nThe successful man will profit from his mistakes and try again in a different way
\nA successful man is one who can lay a firm foundation with the bricks others have thrown at him\nSuccess usu
ally comes to those who are too busy looking for it\nWe cannot solve problems with the kind of thinking we emp
loyed when we came up with them\nJust one small positive thought in the morning can change your whole day\nYou
can get everything in life you want if you will just help enough other people get what they want'
```

Word Counting

Counting



name = ‘baby’

b | a | b | y

{‘b’:2, ‘a’: 1, ‘y’: 1}

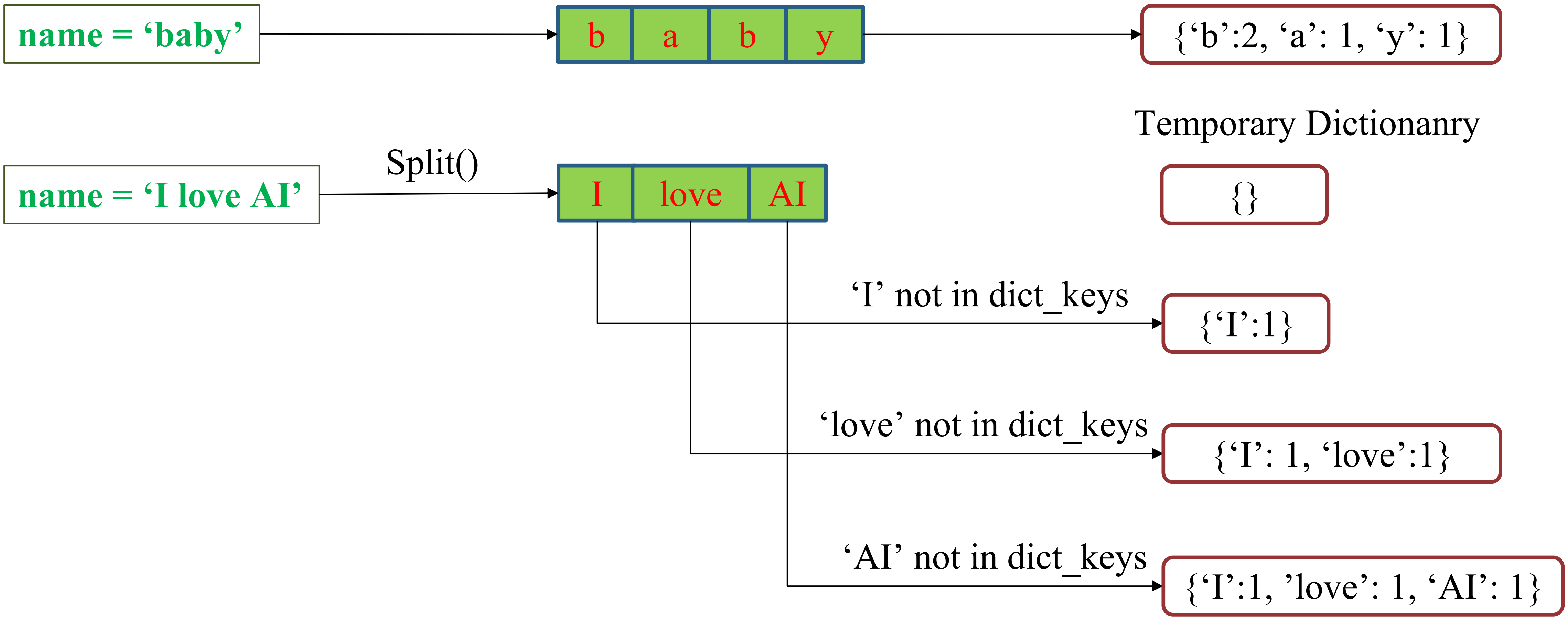
name = ‘I love AI’

Split()

I | love | AI

Word Counting

Counting



name = ‘baby’

b	a	b	y

{‘b’:2, ‘a’: 1, ‘y’: 1}

Temporary Dictionanry

name = ‘I love AI’

Split()

I	love	AI

{}

‘I’ not in dict_keys

{‘I’:1}

‘love’ not in dict_keys

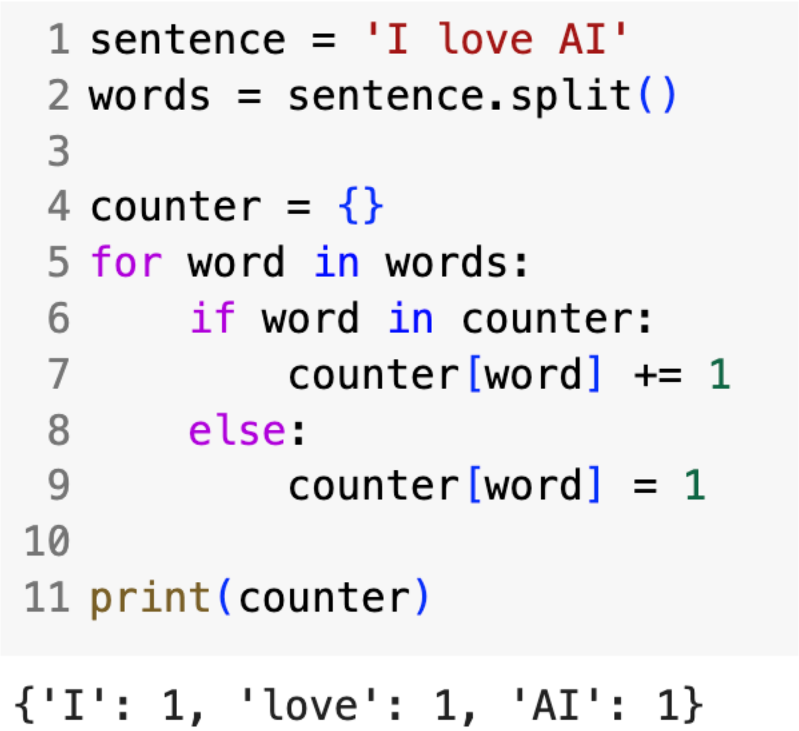
{‘I’: 1, ‘love’:1}

‘AI’ not in dict_keys

{‘I’:1, ’love’: 1, ‘AI’: 1}

Word Counting

Counting

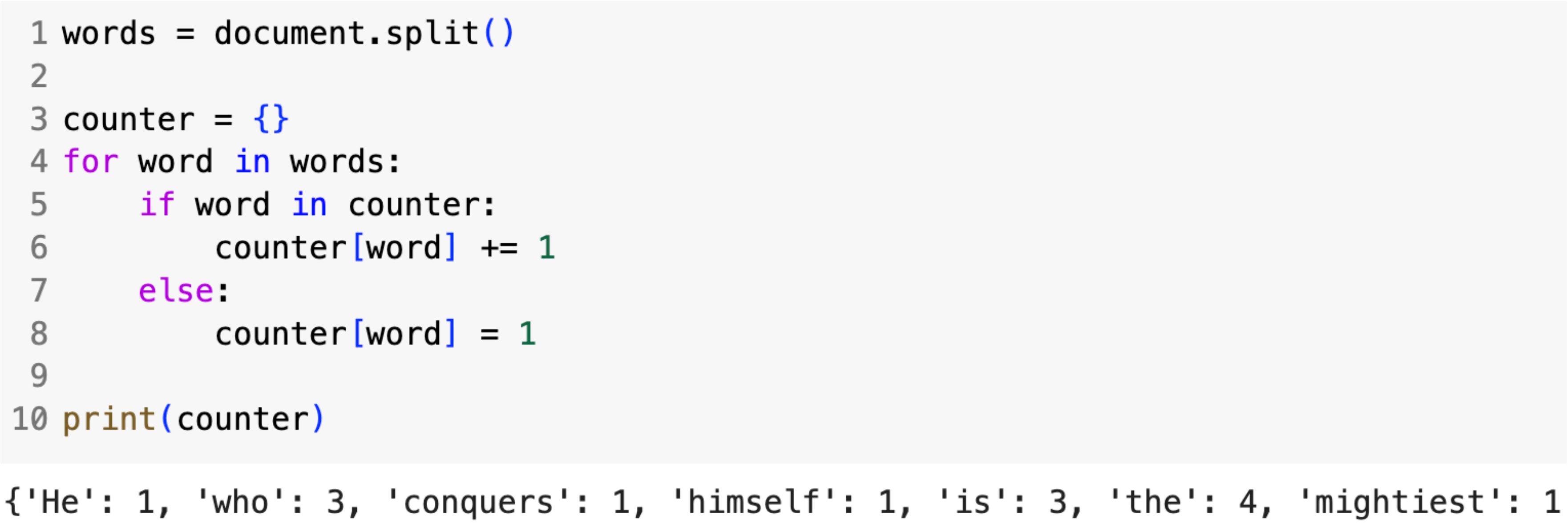


```python
sentence = 'I love AI'
words = sentence.split()

counter = {}
for word in words:
    if word in counter:
        counter[word] += 1
    else:
        counter[word] = 1

print(counter)
```

```
{'I': 1, 'love': 1, 'AI': 1}
```

Word Counting

Word Counting

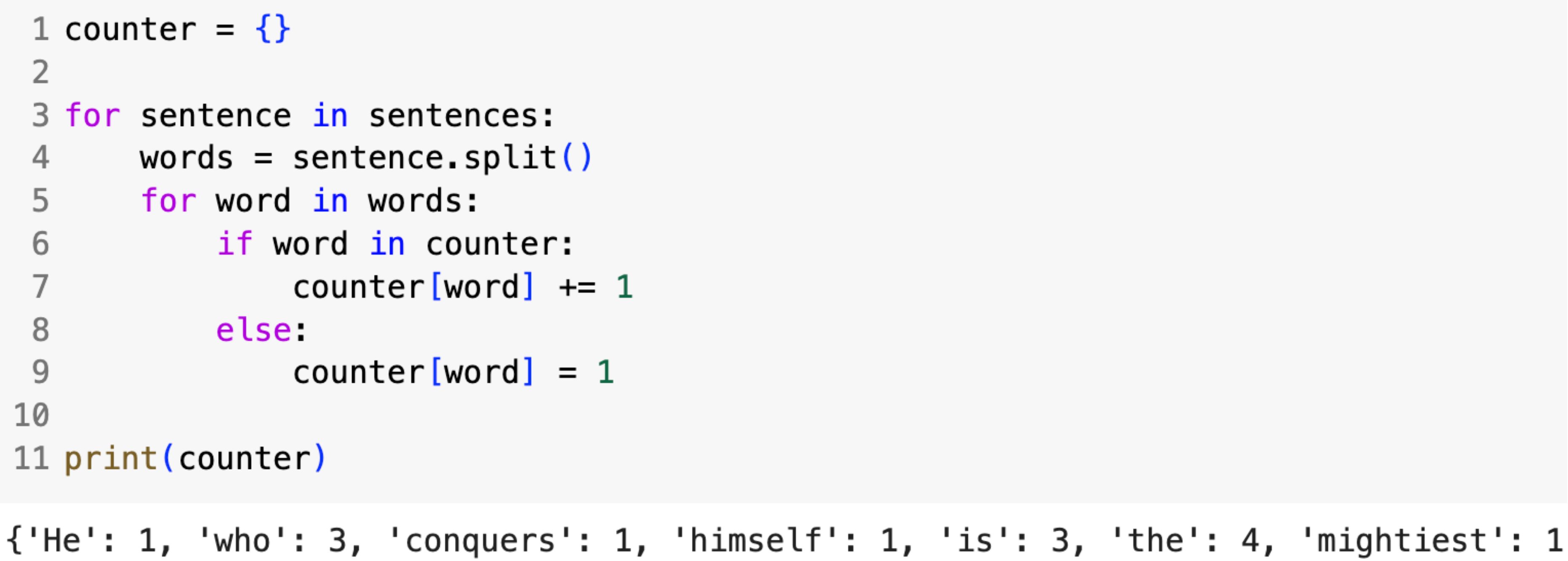


```python
words = document.split()

counter = {}
for word in words:
    if word in counter:
        counter[word] += 1
    else:
        counter[word] = 1

print(counter)
```

```
{'He': 1, 'who': 3, 'conquers': 1, 'himself': 1, 'is': 3, 'the': 4, 'mightiest': 1
```

Word Counting

Word Counting

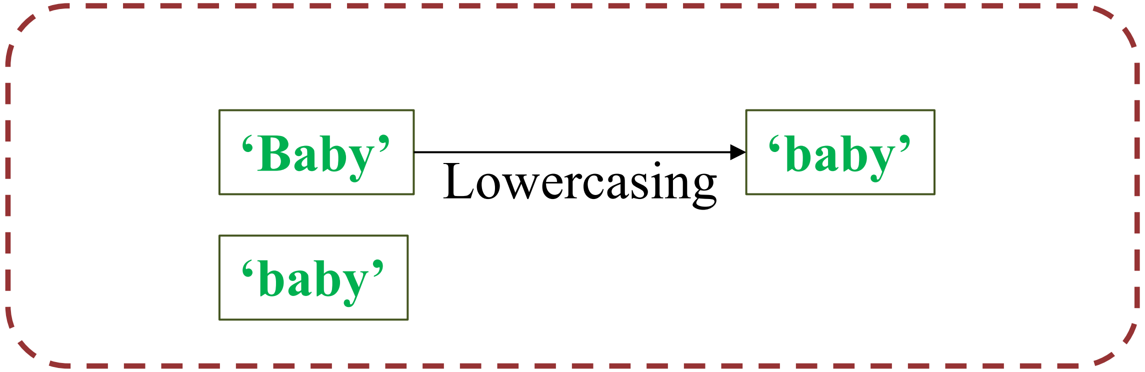


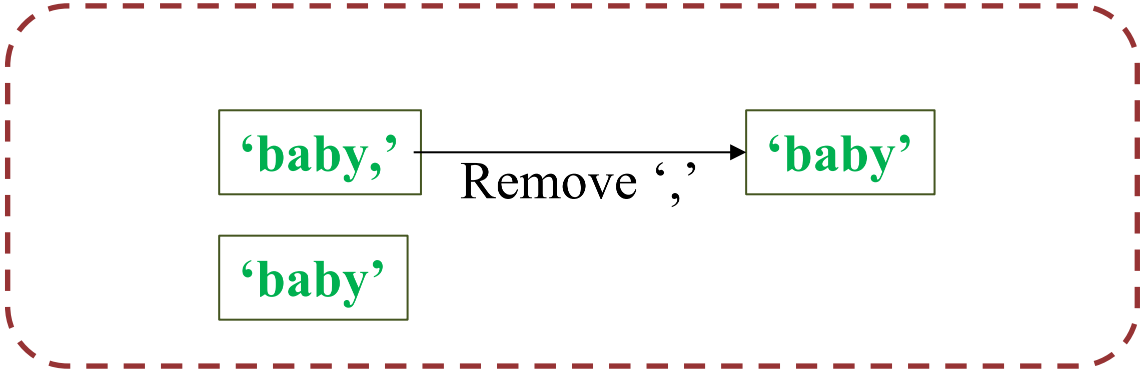
```python
counter = {}

for sentence in sentences:
    words = sentence.split()
    for word in words:
        if word in counter:
            counter[word] += 1
        else:
            counter[word] = 1

print(counter)
```

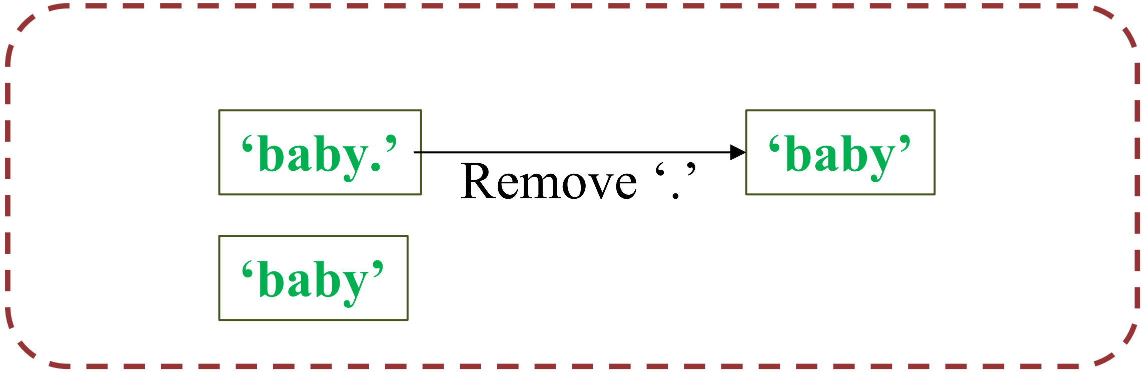
```
{'He': 1, 'who': 3, 'conquers': 1, 'himself': 1, 'is': 3, 'the': 4, 'mightiest': 1
```

Word Counting

Text Preprocessing

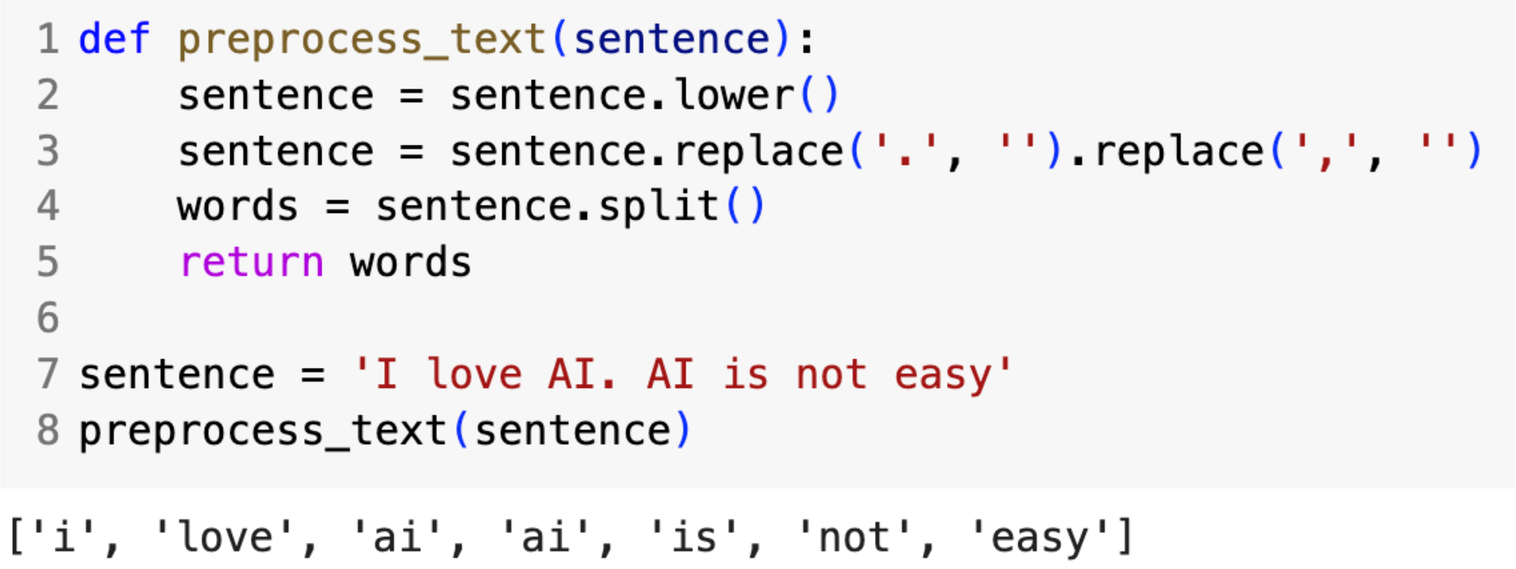






Word Counting

Text Preprocessing

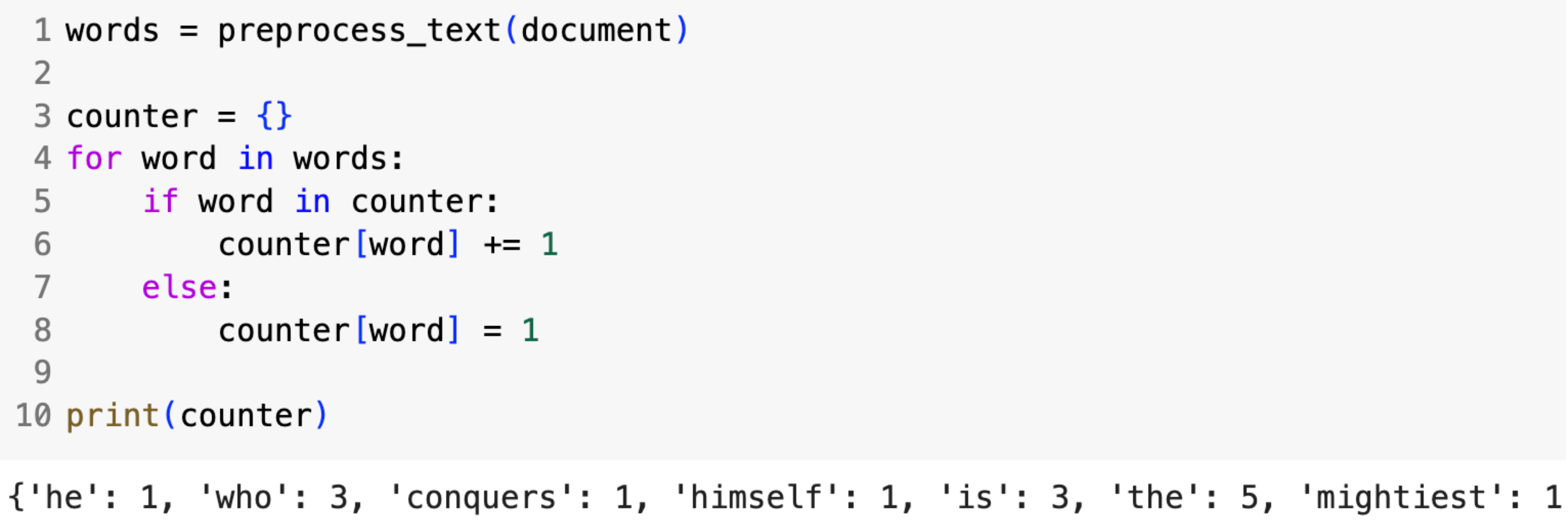


```python
def preprocess_text(sentence):
    sentence = sentence.lower()
    sentence = sentence.replace('.', '').replace(',', '')
    words = sentence.split()
    return words

sentence = 'I love AI. AI is not easy'
preprocess_text(sentence)
```

```
['i', 'love', 'ai', 'ai', 'is', 'not', 'easy']
```

Word Counting

Text Preprocessing

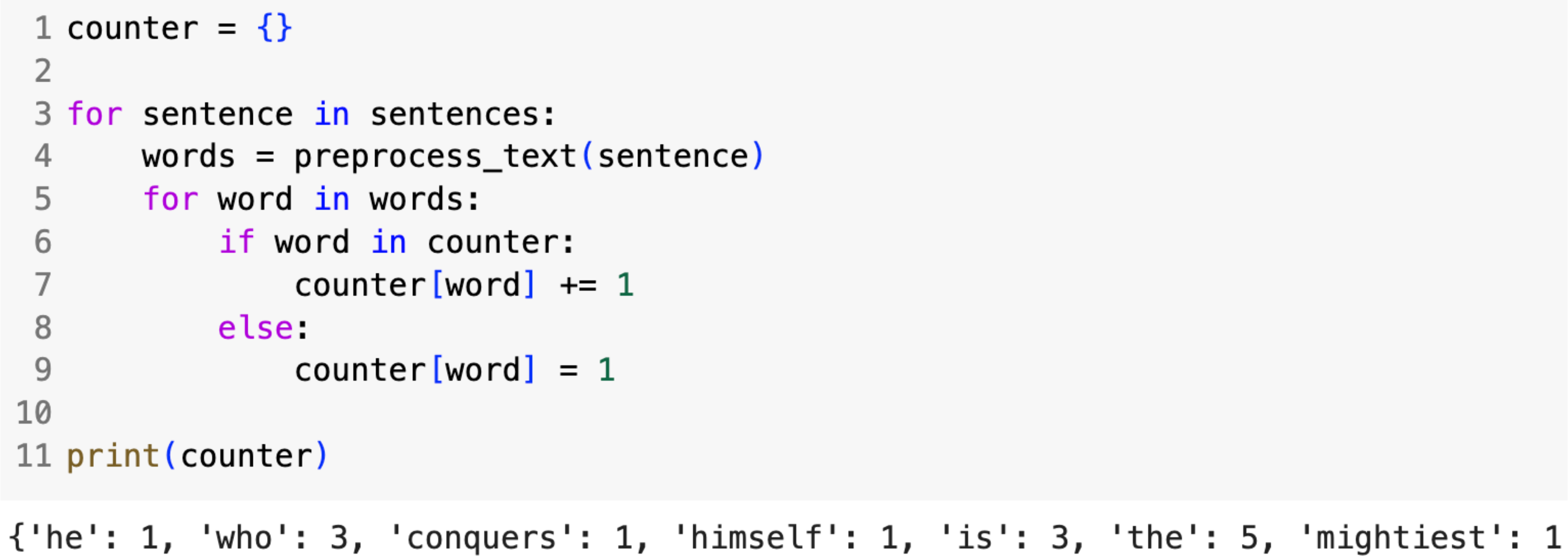


```python
words = preprocess_text(document)

counter = {}
for word in words:
    if word in counter:
        counter[word] += 1
    else:
        counter[word] = 1

print(counter)
```

```
{'he': 1, 'who': 3, 'conquers': 1, 'himself': 1, 'is': 3, 'the': 5, 'mightiest': 1
```

Word Counting

Text Preprocessing



```python
counter = {}

for sentence in sentences:
    words = preprocess_text(sentence)
    for word in words:
        if word in counter:
            counter[word] += 1
        else:
            counter[word] = 1

print(counter)
```

```
{'he': 1, 'who': 3, 'conquers': 1, 'himself': 1, 'is': 3, 'the': 5, 'mightiest': 1
```

Outline

SECTION 1
Getting Max Over Kernel

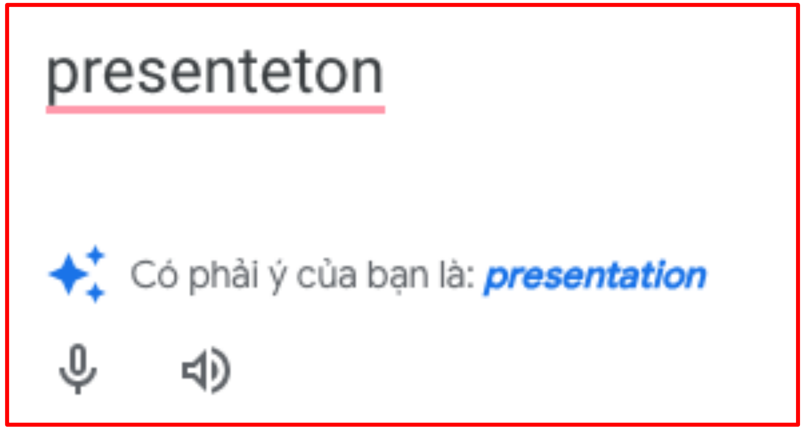
SECTION 2
Character Counting

SECTION 3
Word Counting (File)

SECTION 4
Levenshtein Distance

Levenshtein Distance

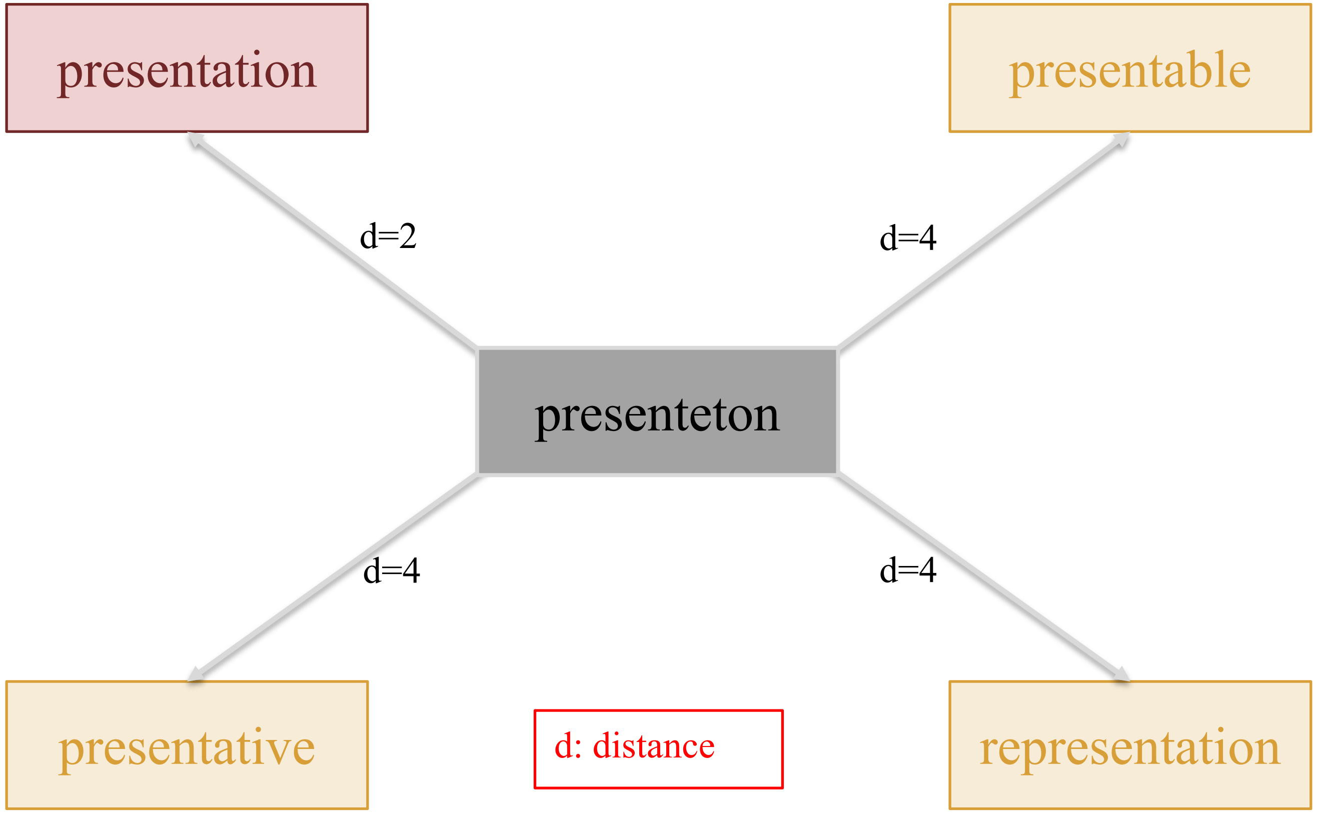
Getting Started



presenteton

Có phải ý của bạn là: ***presentation***

How to measure the similarity or gap between two strings ?

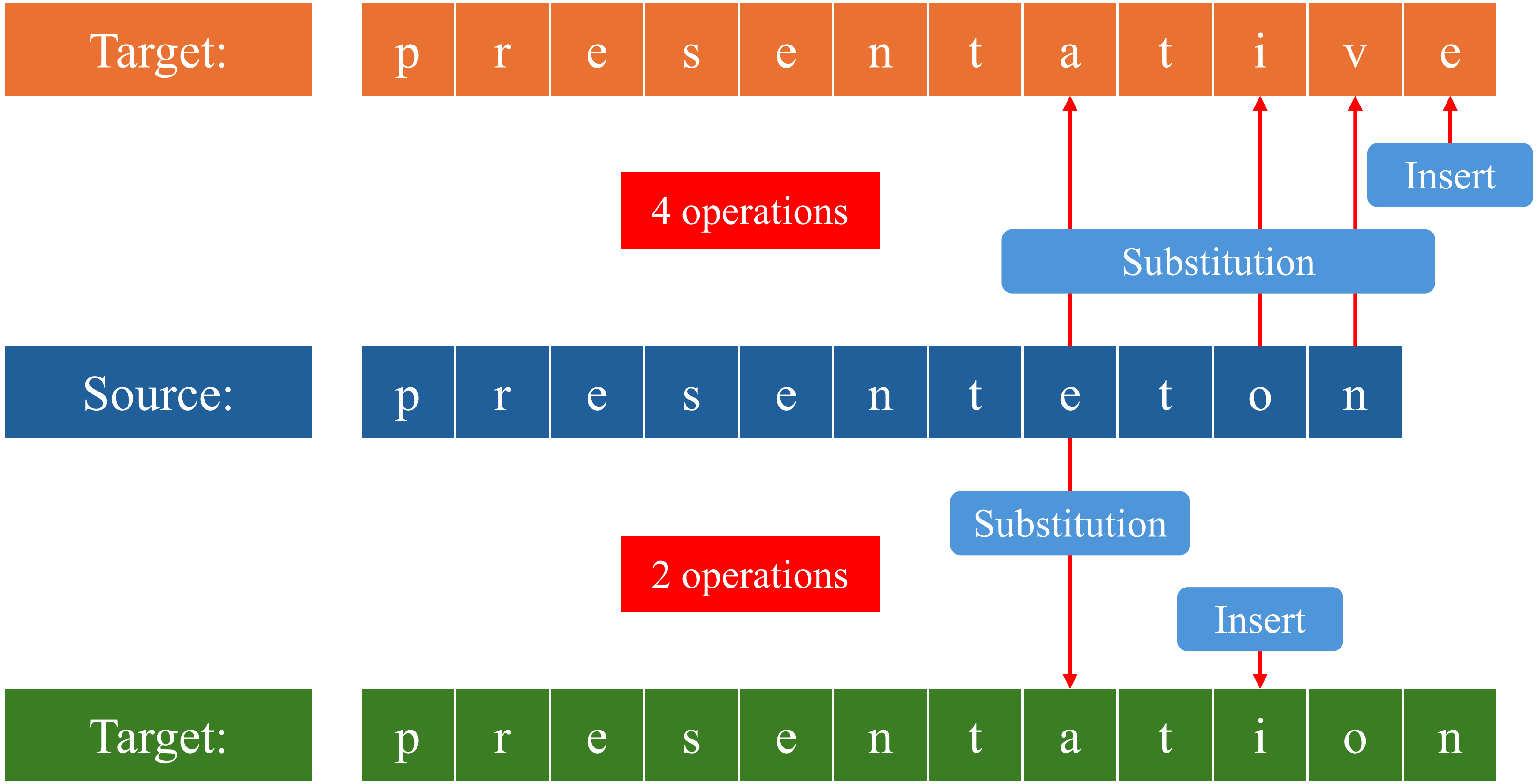


Levenshtein Distance

Minimum number of editing operations

Operations

- Delete
- Insert
- Substitution



Target:	p	r	e	s	e	n	t	a	t	i	v	e
Source:	p	r	e	s	e	n	t	e	t	o	n	
Target:	p	r	e	s	e	n	t	a	t	i	o	n

Insert

4 operations

Substitution

Substitution

2 operations

Insert

Levenshtein Distance

The Minimum Edit Distance Algorithm

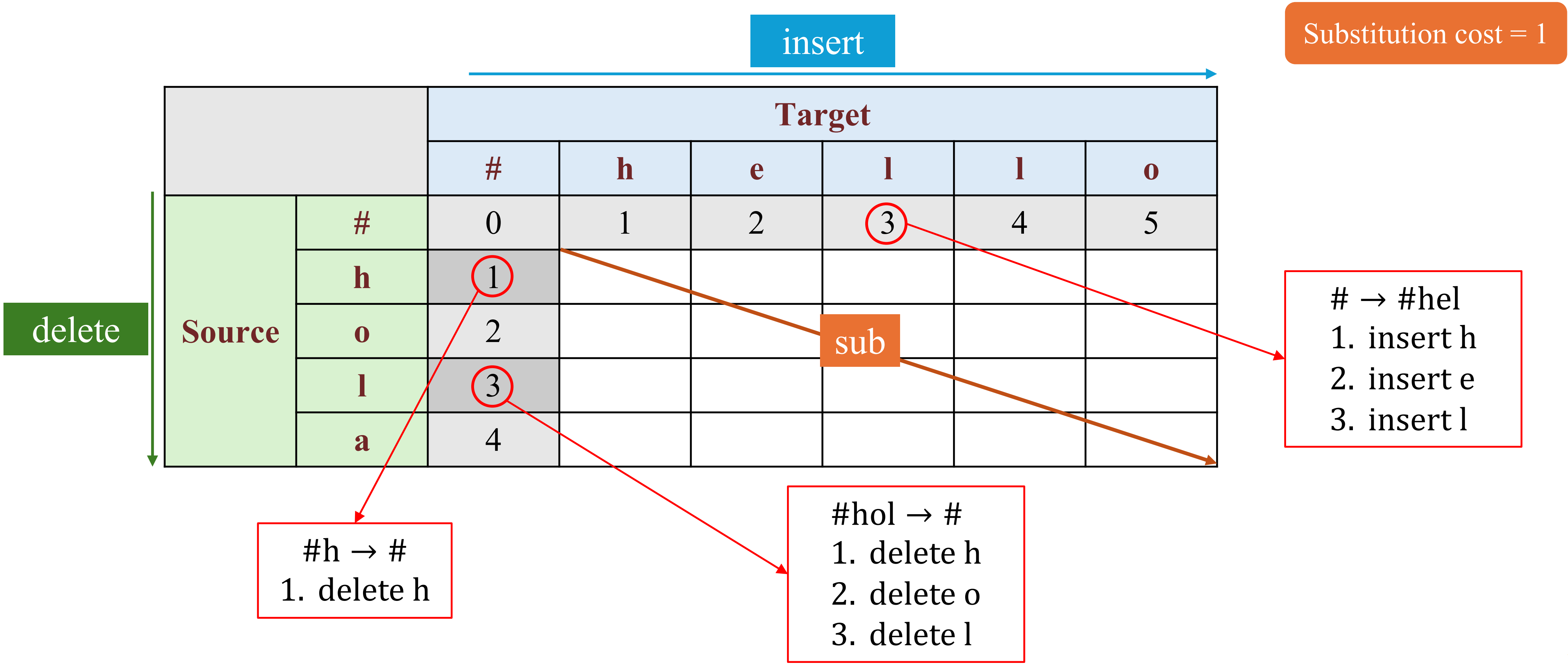
Source: hola

Target: hello

Delete cost = 1

Insert cost = 1

Substitution cost = 1



		Target					
		#	h	e	l	l	o
Source	#	0	1	2	3	4	5
	h	1					
	o	2					
	l	3					
	a	4					

# → #hel
1. insert h
2. insert e
3. insert l

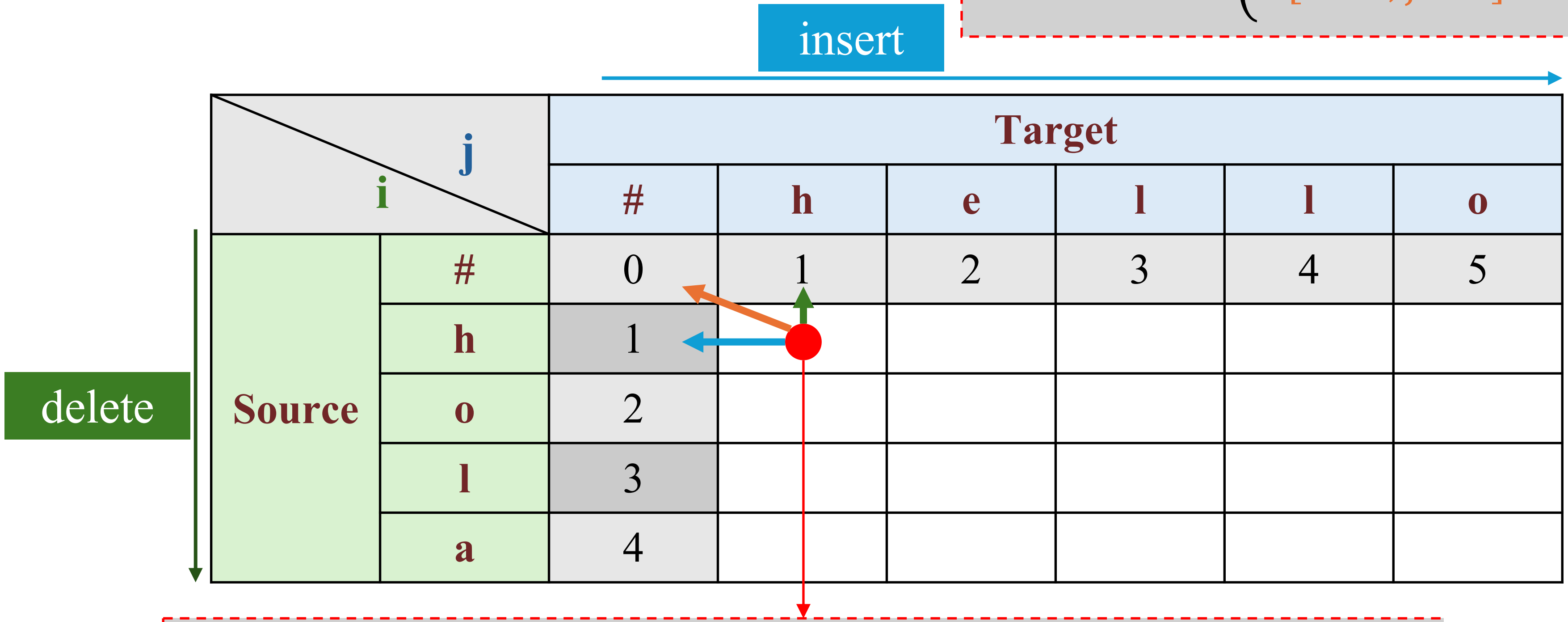
#h → #
1. delete h

#hol → #
1. delete h
2. delete o
3. delete l

Levenshtein Distance

! Algorithm

$$D[i,j] = min \begin{cases} \mathrm{D[i-1,j] + delcost(source[i])} \\ \mathrm{D[i,j-1] + inscost(target[j])} \\ \mathrm{D[i-1,j-1] + subcost(source[i], target[j])} \end{cases}$$


insert
delete

	j i	Target					
		#	h	e	l	l	o
Source	#	0	1	2	3	4	5
	h	1					
	o	2					
	l	3					
	a	4					

Delete cost = 1

Insert cost = 1

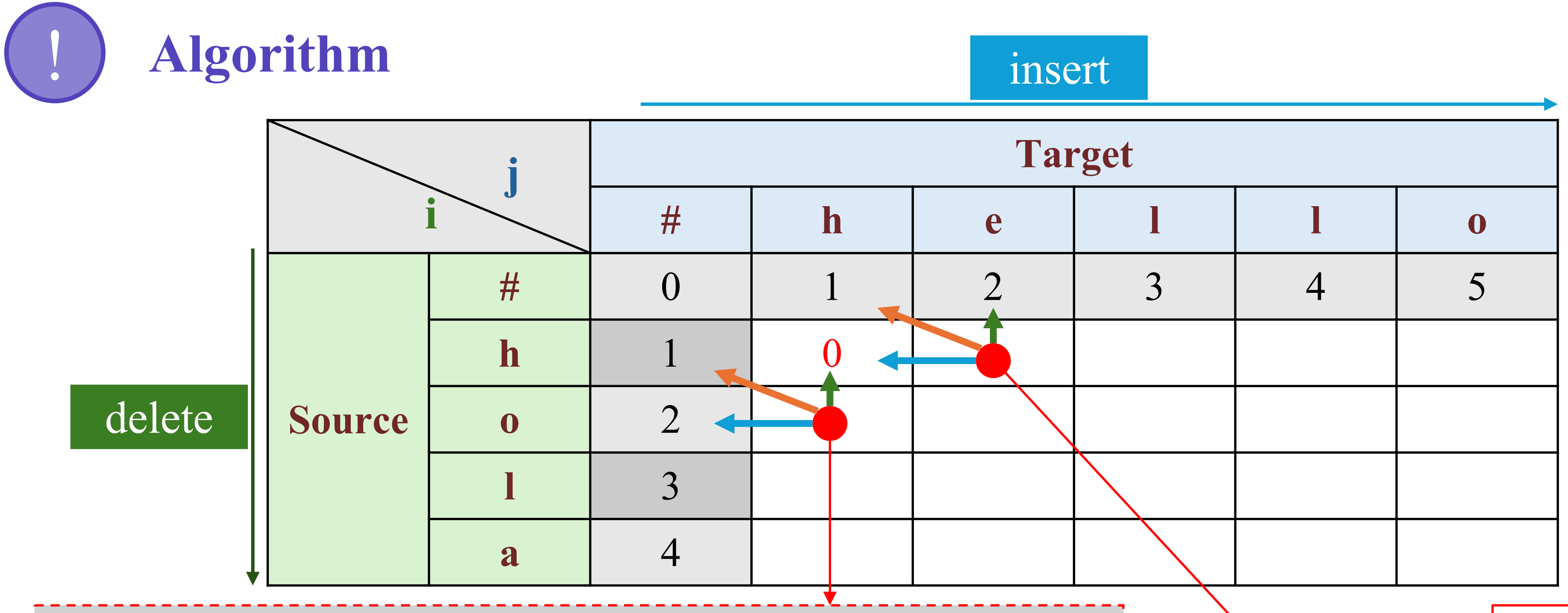
Substitution cost = 1

$$D[1,1] = min \begin{cases} \mathrm{D[0,1] + delcost(source[1]) = 1 + 1 = 2} \\ \mathrm{D[1,0] + inscost(}target\mathrm{[1]) = 1 + 1 = 2} \\ \mathrm{D[0,0] + subcost(}source\mathrm{[1], target[1]) = 0 + 0 = 0} \end{cases}$$

Note:
If source[i] == target[j]:
subcost(source[i] ,target[j])=0

Levenshtein Distance

Algorithm



i \ j		Target					
		#	h	e	l	l	o
Source	#	0	1	2	3	4	5
	h	1	0				
	o	2					
	l	3					
	a	4					

Delete cost = 1

Insert cost = 1

Substitution cost = 1

$$D[2,1] = min \begin{cases} \mathrm{D}[1,1] + \mathrm{delcost}(\mathrm{source}[2]) = 0 + 1 = 1 \\ \mathrm{D}[2,0] + \mathrm{inscost}(\mathrm{target}[1]) = 2 + 1 = 3 \\ \mathrm{D}[1,0] + \mathrm{subcost}(source[2], \mathrm{target}[1]) = 1 + 1 = 2 \end{cases}$$

Note:
If source[i] == target[j]:
subcost(source[i] ,target[j])=0

$$D[1,2] = min \begin{cases} \mathrm{D}[0,1] + \mathrm{delcost}(\mathrm{source}[1]) = 2 + 1 = 3 \\ \mathrm{D}[1,1] + \mathrm{inscost}(\mathrm{target}[2]) = 0 + 1 = 1 \\ \mathrm{D}[0,1] + \mathrm{subcost}(source[1], \mathrm{target}[2]) = 1 + 1 = 2 \end{cases}$$

Levenshtein Distance

Algorithm

insert

delete

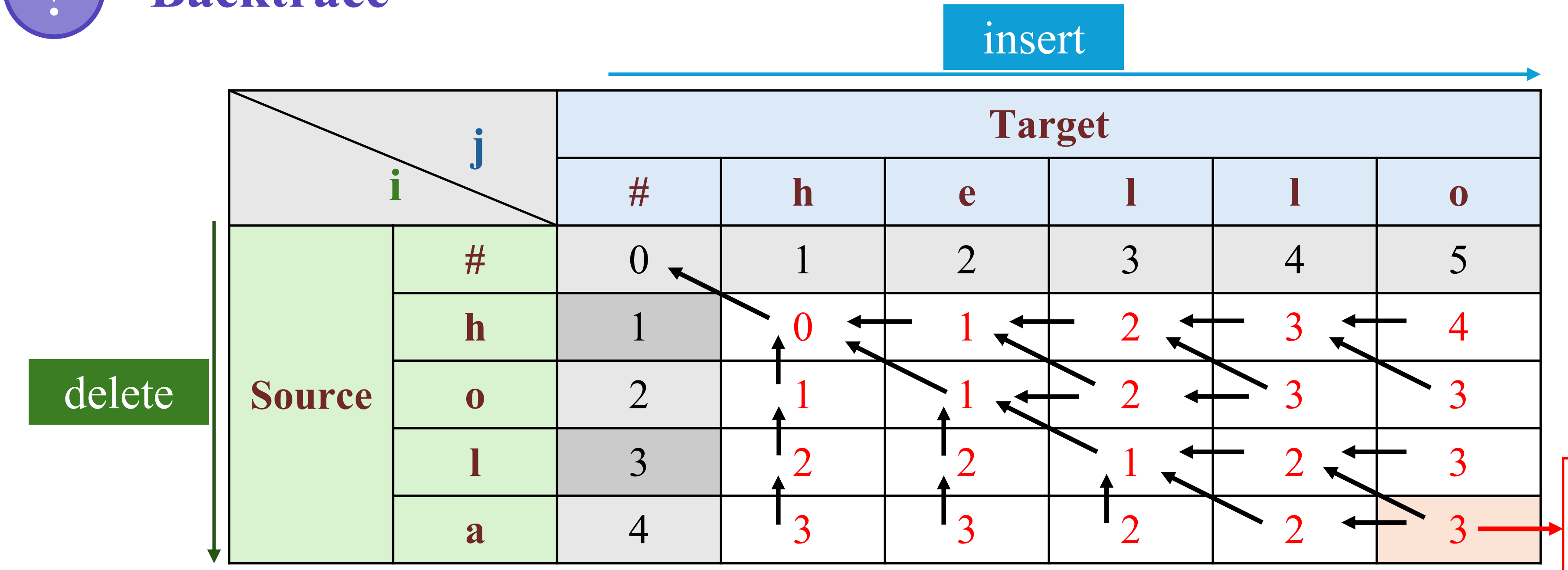
j i		Target					
		#	h	e	l	l	o
Source	#	0	1	2	3	4	5
	h	1	0	1	2	3	4
	o	2	1	1	2	3	3
	l	3	2	2	1	2	3
	a	4	3	3	2	2	3

D[i,j] = D[4, 5]
= editDistance(“hola”,“hello”)
= 3

When going down each step, store back pointers in each cell to serve for the backtrace phase.

Levenshtein Distance

Backtrace



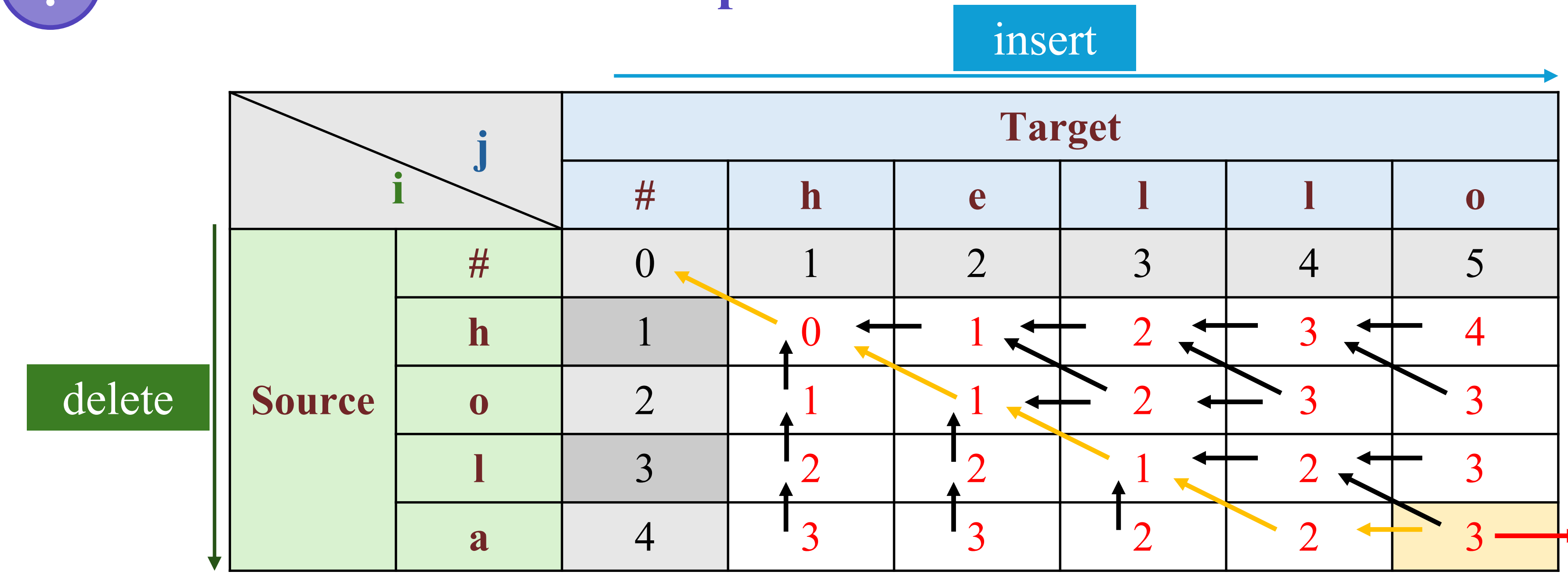
i \ j		#	h	e	l	l	o
Source	#	0	1	2	3	4	5
	h	1	0	1	2	3	4
	o	2	1	1	2	3	3
	l	3	2	2	1	2	3
	a	4	3	3	2	2	3

D[i,j] = D[4, 5]
= editDistance(“hola”, “hello”)
= 3

- Starting from the last cell and returning based on choosing the minimum cell value.
- Each cell may have multiple to return to because they have the same minimum value.

Levenshtein Distance

Minimum edit distance path



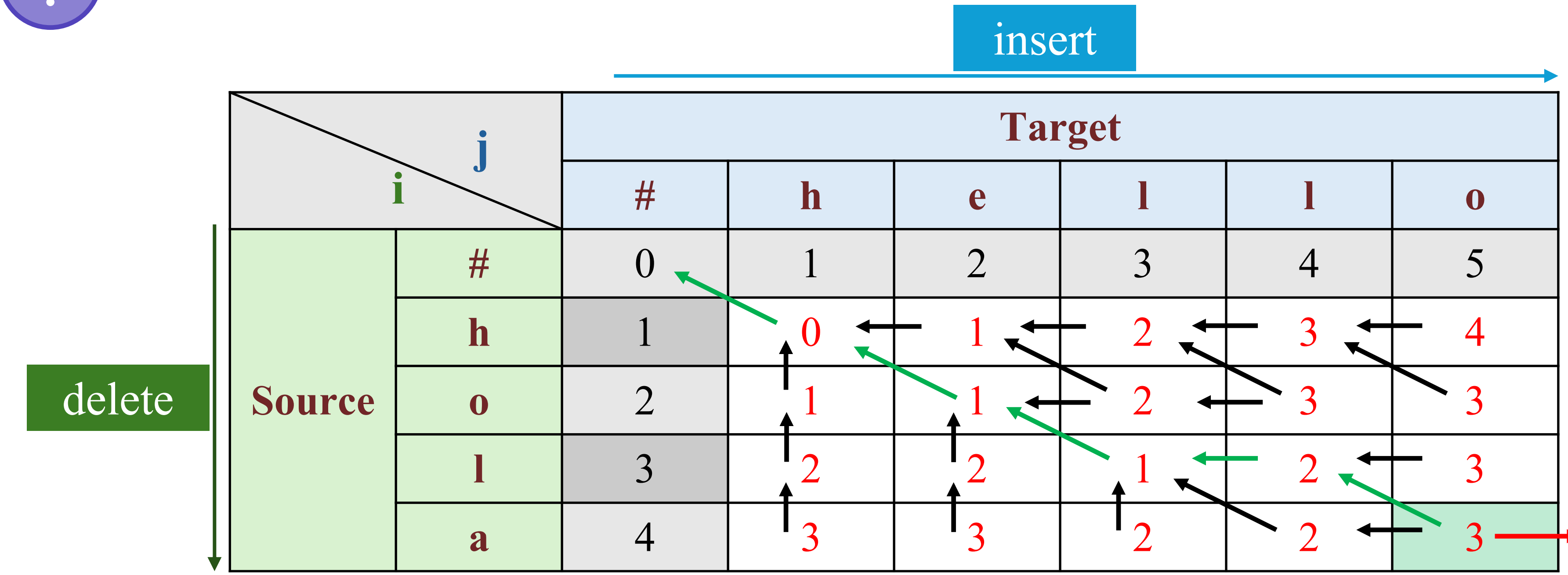
j i		Target					
		#	h	e	l	l	o
Source	#	0	1	2	3	4	5
	h	1	0	1	2	3	4
	o	2	1	1	2	3	3
	l	3	2	2	1	2	3
	a	4	3	3	2	2	3

D[i,j] = D[4, 5]
= editDistance(“hola”, “hello”)
= 3

- This is one of the minimum edit distance paths.
- Modify steps (going in the reverse direction with the backtrace path):
  - sub(h, h) => hola; cost = 0
  - sub(o, e) => hela; cost = 1
  - sub(l, l) => hela; cost = 0
  - sub(a, l) => hell; cost = 1
  - insert(o) => hello; cost = 1

Levenshtein Distance

Backtrace



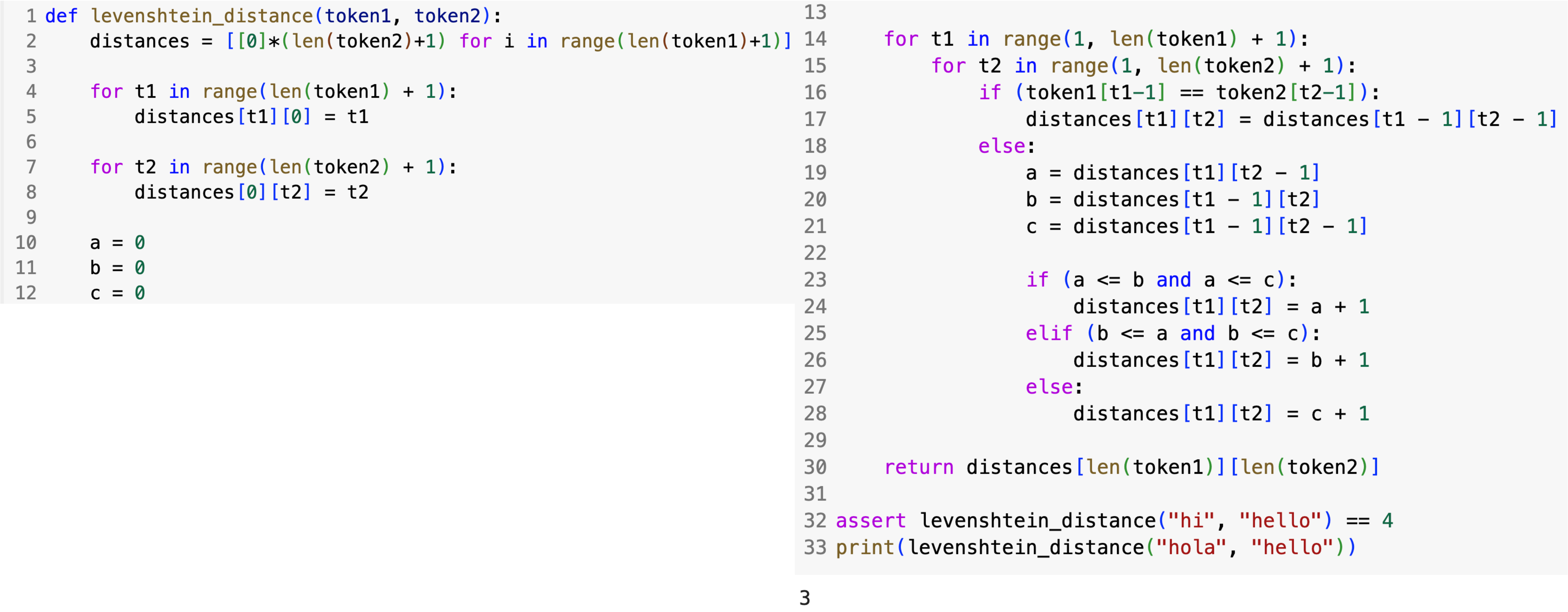
i \ j	#	h	e	l	l	o
#	0	1	2	3	4	5
h	1	0	1	2	3	4
o	2	1	1	2	3	3
l	3	2	2	1	2	3
a	4	3	3	2	2	3

D[i,j] = D[4, 5]
= editDistance("hola", "hello")
= 3

- This is another minimum edit distance path.
- Modify steps (going in the reverse direction with the backtrace path):
  - sub(h, h) => hola; cost = 0
  - sub(o, e) => hela; cost = 1
  - sub(l, l) => hela; cost = 0
  - insert(l) => hella; cost = 1
  - sub(a, o) => hello; cost = 1

Levenshtein Distance

Code

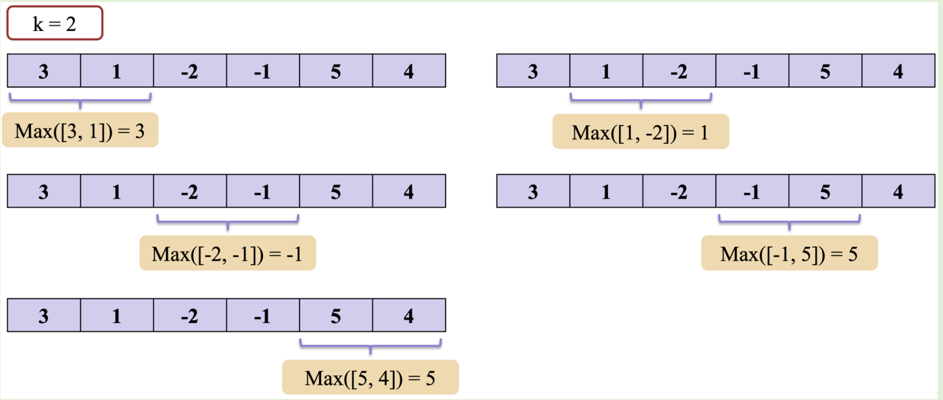


```python
def levenshtein_distance(token1, token2):
    distances = [[0]*(len(token2)+1) for i in range(len(token1)+1)]

    for t1 in range(len(token1) + 1):
        distances[t1][0] = t1

    for t2 in range(len(token2) + 1):
        distances[0][t2] = t2

    a = 0
    b = 0
    c = 0

    for t1 in range(1, len(token1) + 1):
        for t2 in range(1, len(token2) + 1):
            if (token1[t1-1] == token2[t2-1]):
                distances[t1][t2] = distances[t1 - 1][t2 - 1]
            else:
                a = distances[t1][t2 - 1]
                b = distances[t1 - 1][t2]
                c = distances[t1 - 1][t2 - 1]

                if (a <= b and a <= c):
                    distances[t1][t2] = a + 1
                elif (b <= a and b <= c):
                    distances[t1][t2] = b + 1
                else:
                    distances[t1][t2] = c + 1

    return distances[len(token1)][len(token2)]

assert levenshtein_distance("hi", "hello") == 4
print(levenshtein_distance("hola", "hello"))
```

```
3
```

Summary

Getting Max Over Kernel



k = 2

3 | 1 | -2 | -1 | 5 | 4

Max([3, 1]) = 3

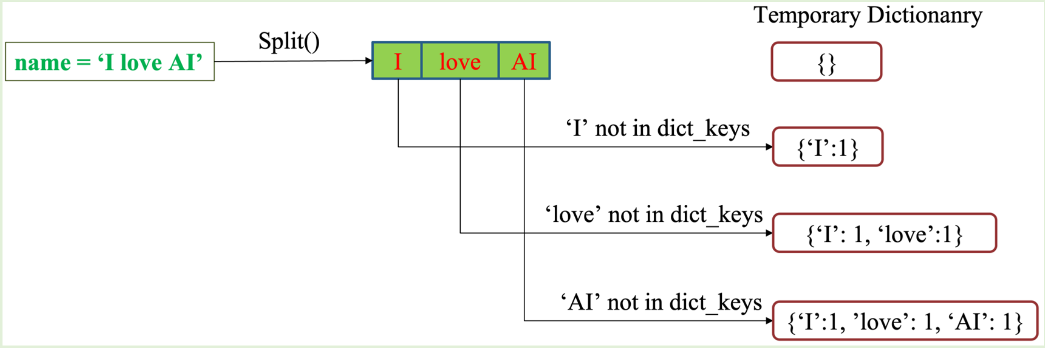
Max([1, -2]) = 1

Max([-2, -1]) = -1

Max([-1, 5]) = 5

Max([5, 4]) = 5

Word Counting



name = ‘I love AI’

Split()

I | love | AI

Temporary Dictionanry

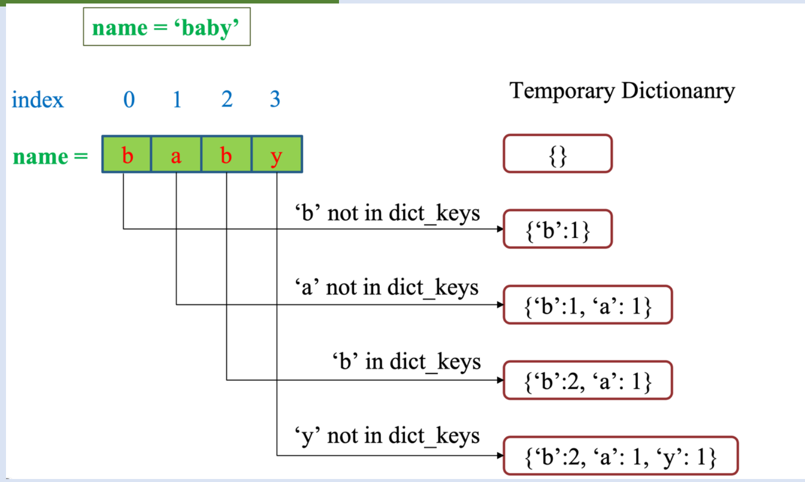
{}

‘I’ not in dict_keys → {‘I’:1}

‘love’ not in dict_keys → {‘I’: 1, ‘love’:1}

‘AI’ not in dict_keys → {‘I’:1, ’love’: 1, ‘AI’: 1}

Character Counting



name = ‘baby’

index 0 1 2 3

name = b | a | b | y

Temporary Dictionanry

{}

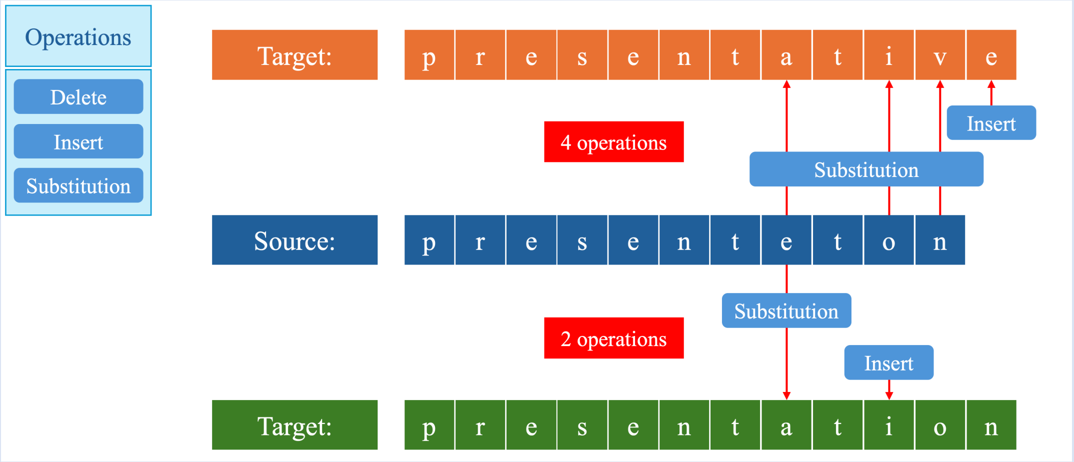
‘b’ not in dict_keys → {‘b’:1}

‘a’ not in dict_keys → {‘b’:1, ‘a’: 1}

‘b’ in dict_keys → {‘b’:2, ‘a’: 1}

‘y’ not in dict_keys → {‘b’:2, ‘a’: 1, ‘y’: 1}

Levenshtein



Operations

- Delete
- Insert
- Substitution

Target: p r e s e n t a t i v e

4 operations

Substitution

Insert

Source: p r e s e n t e t o n

2 operations

Substitution

Insert

Target: p r e s e n t a t i o n

Thanks!

Any questions?