Chapter 8
Strings and Regular Expressions

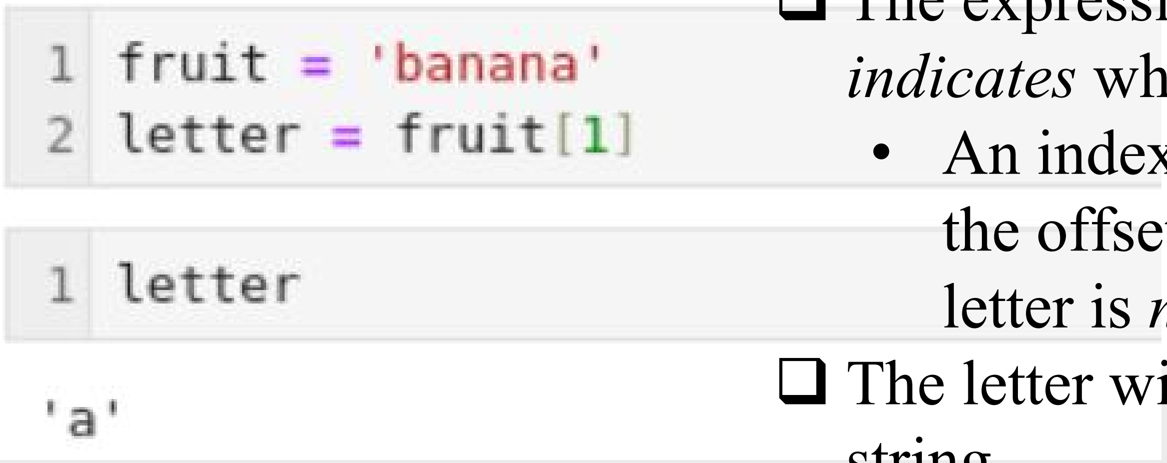
A. Downey, *Think Python: How to Think Like a Computer Science,* 3rd ed., O'Reilly, 2024.

https://allendowney.github.io/ThinkPython/

Contents

1. A String Is a Sequence

- A string is a sequence of characters. A **character** can be a letter (in almost any alphabet), a digit, a punctuation mark, or whitespace.
- You can select a character from a string with the bracket operator. This example statement selects character number 1 (the second character) from **fruit** and assigns it to **letter**:



```
fruit = 'banana'
letter = fruit[1]
```

```
letter
```

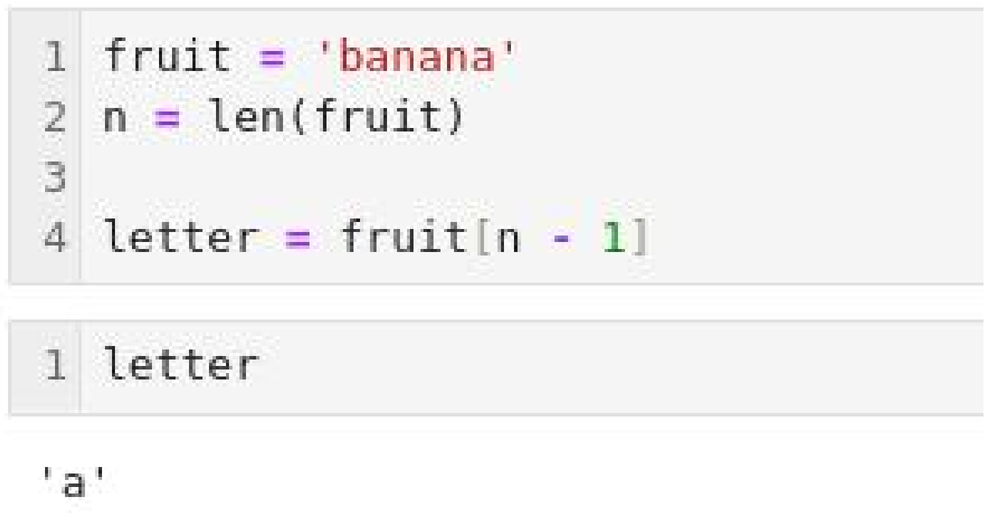
```
'a'
```

❑ The expression in brackets is an **index**, so called because it *indicates* which character in the sequence to select.
- An index is an offset from the beginning of the string, so the offset of the first letter is 0. The offset of the last letter is n - 1, where n is the length of the string.

❑ The letter with index 1 is actually the second letter of the string.

1. A String Is a Sequence

- To get the last letter of a string

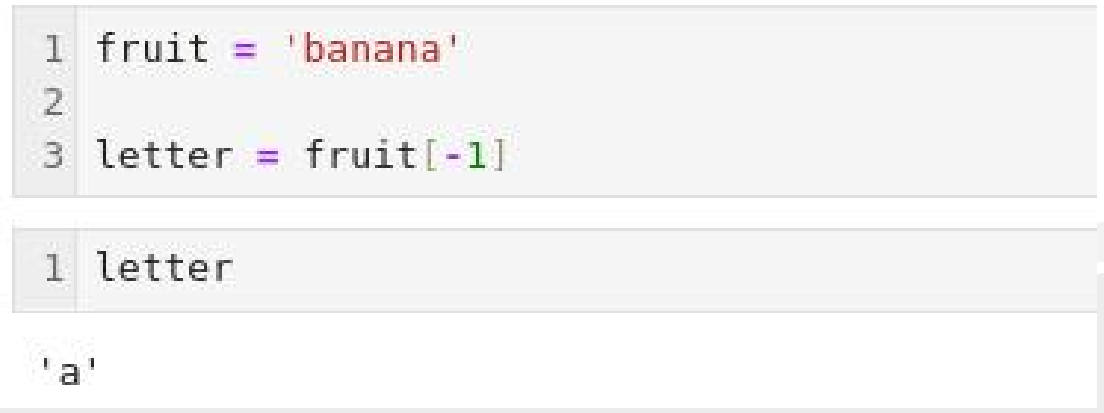


```
fruit = 'banana'
n = len(fruit)

letter = fruit[n - 1]
```

```
letter
```

```
'a'
```

- To get the last letter in a string, you can use a negative index, which counts backward from the end.
  - The index -1 selects the last letter, -2 selects the second to last, and so on.

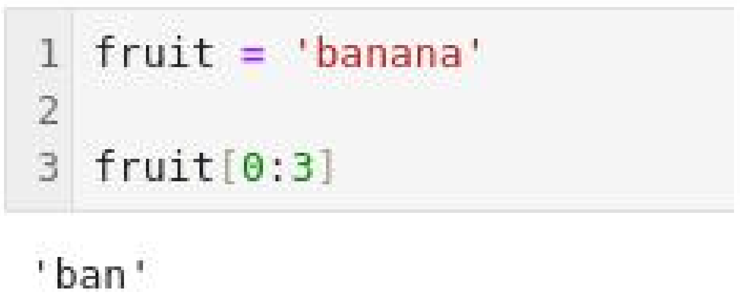


```
fruit = 'banana'

letter = fruit[-1]
```

```
letter
```

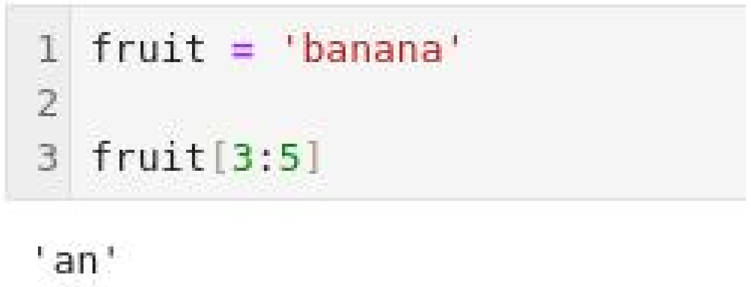
```
'a'
```

2. String Slices

- A segment of a string is called a **slice**. Selecting a slice is similar to selecting a character:
- 

```
fruit = 'banana'

fruit[0:3]
```

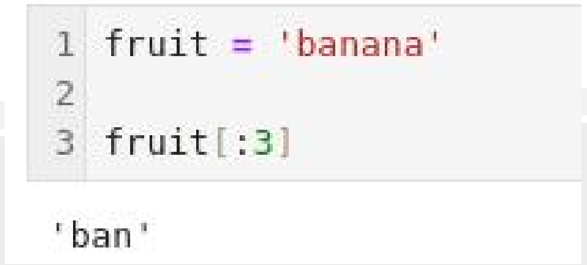
```
'ban'
```



```
fruit = 'banana'

fruit[3:5]
```

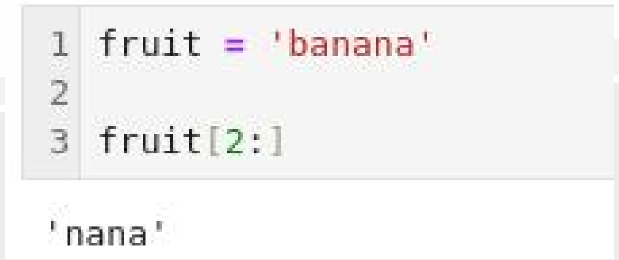
```
'an'
```

  – The operator [*n*:*m*] returns the part of the string from the *n*th character to the *m*th character, including the first but excluding the second.
  – If you omit the first index, the slice starts at the beginning of the string. If you omit the second index, the slice goes to the end of the string:



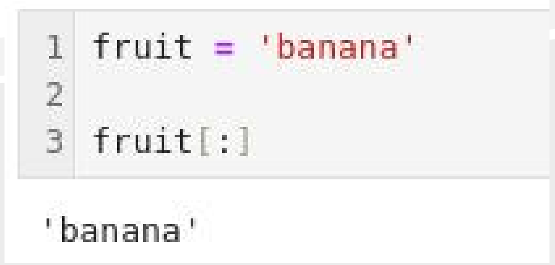
```
fruit = 'banana'

fruit[:3]
```

```
'ban'
```



```
fruit = 'banana'

fruit[2:]
```

```
'nana'
```

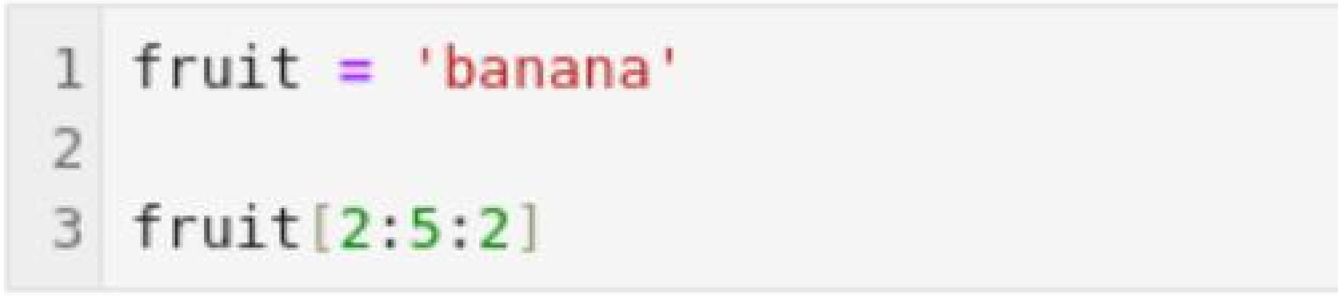


```
fruit = 'banana'

fruit[:]
```

```
'banana'
```

2. String Slices

- The operator [*n*:*m*:*k*] returns the part of the string from the *n*th character to the *m*th character, including the first but excluding the second. The number *k* specifies the step of the slicing (default value of *k* is 1)

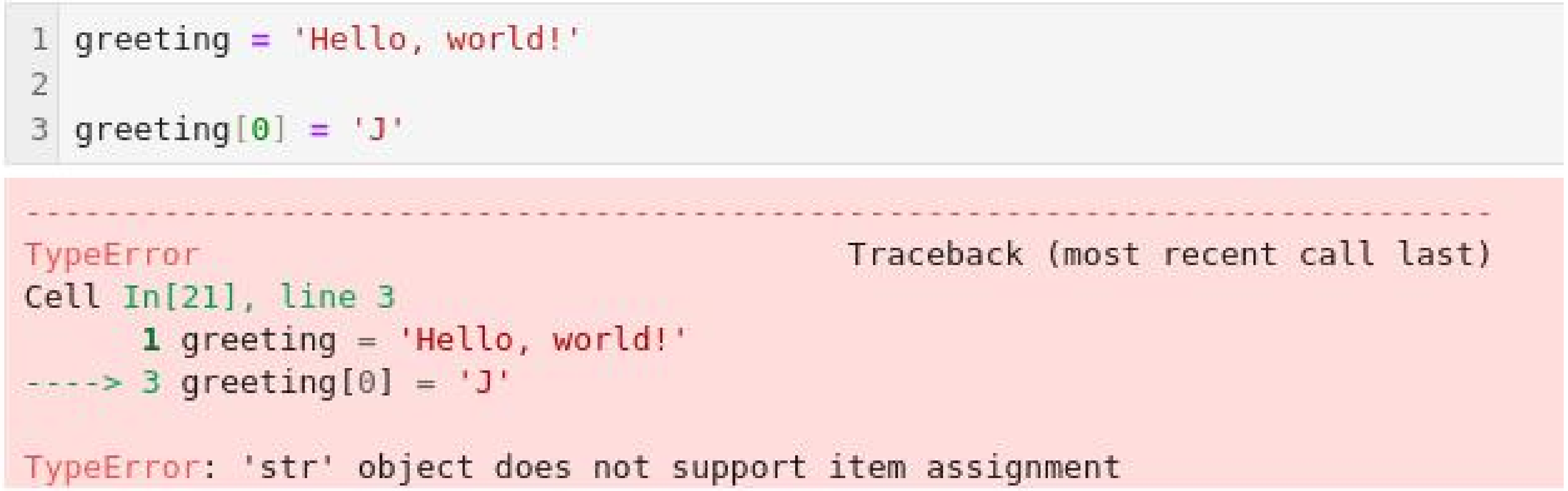


```python
fruit = 'banana'

fruit[2:5:2]
```

```
'nn'
```

3. Strings Are Immutable

- Cannot change a string:

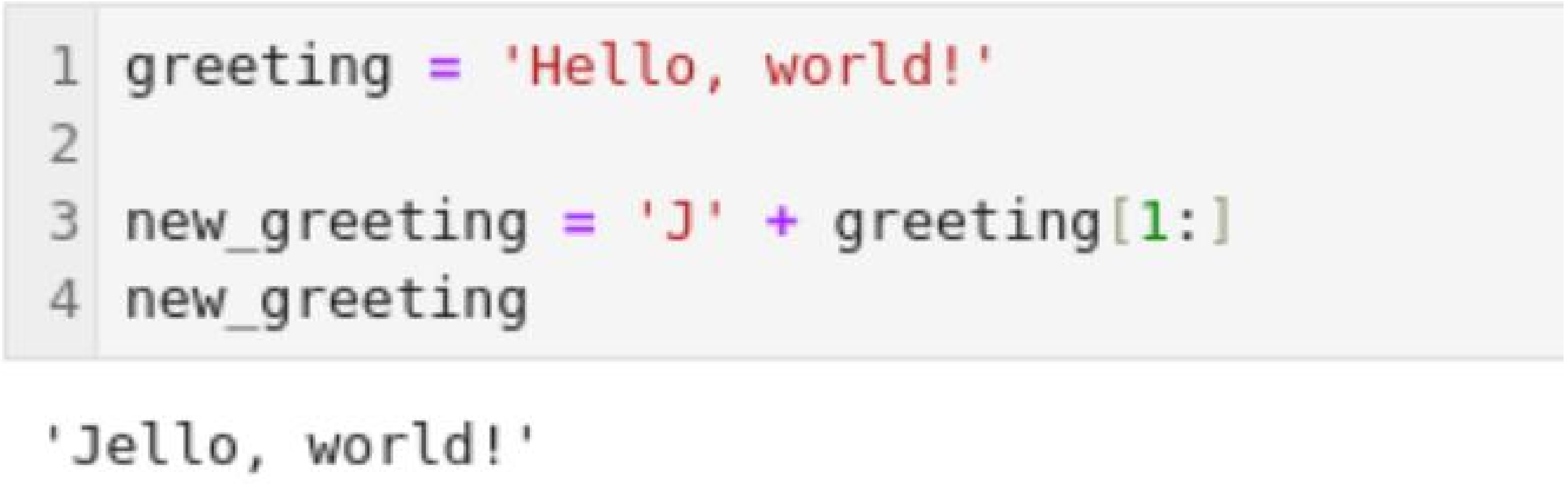


```
greeting = 'Hello, world!'

greeting[0] = 'J'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[21], line 3
      1 greeting = 'Hello, world!'
----> 3 greeting[0] = 'J'

TypeError: 'str' object does not support item assignment
```

  – The reason for this error is that strings are **immutable**, which means you can’t change an existing string.

3. Strings Are Immutable



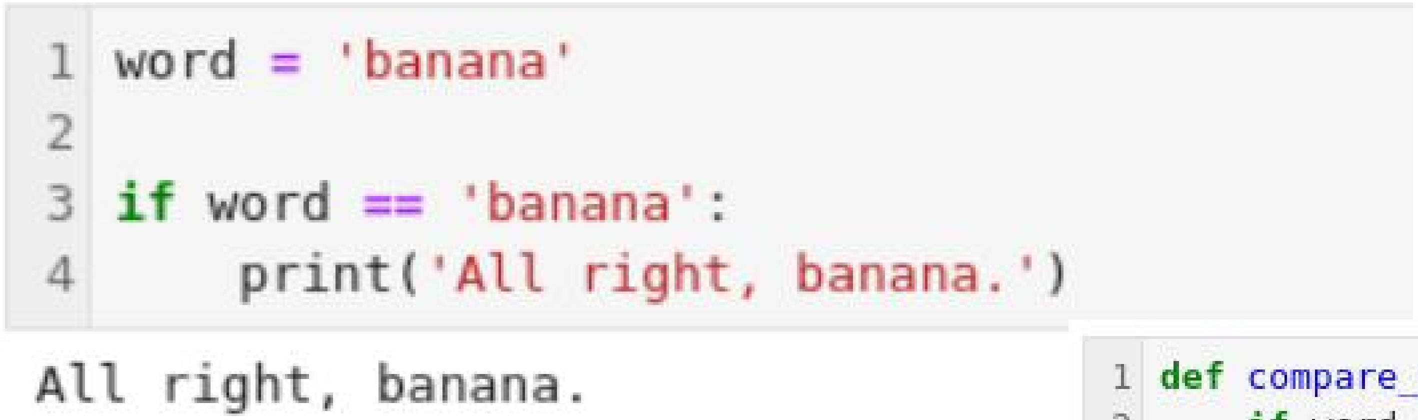
```python
greeting = 'Hello, world!'

new_greeting = 'J' + greeting[1:]
new_greeting
```

```
'Jello, world!'
```

– **new_string** is a new string.

4. String Comparison

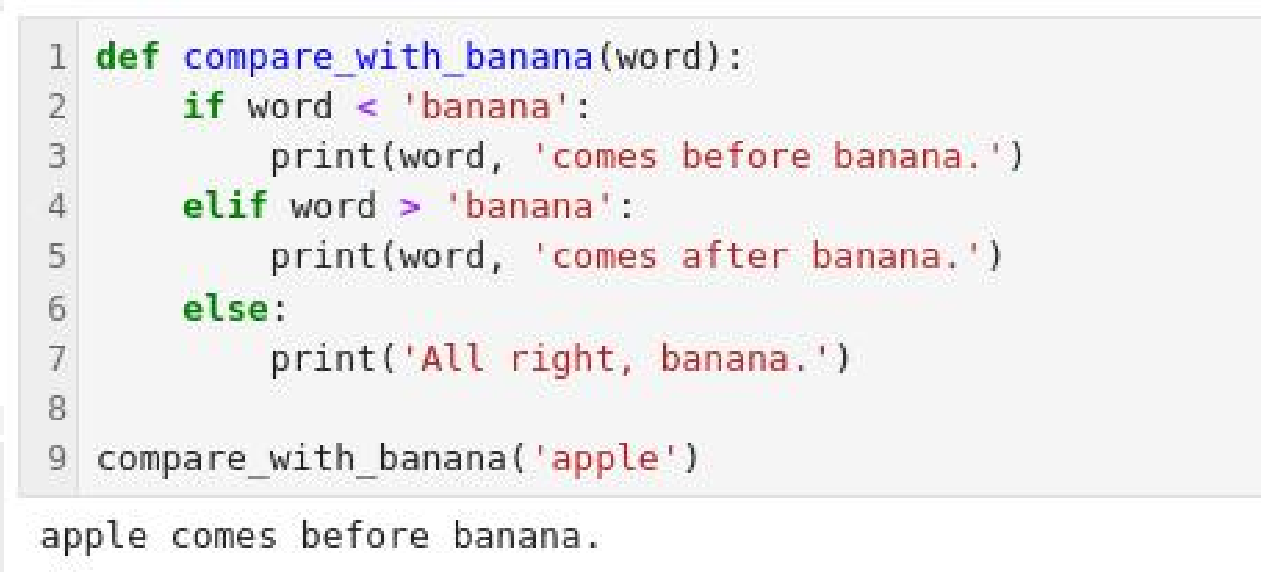
- The relational operators work on strings. To see if two strings are equal, we can use the == operator:



```
word = 'banana'

if word == 'banana':
    print('All right, banana.')
```

```
All right, banana.
```

- Other relational operations (>, <. ≥, ≤) are useful for putting words in alphabetical order.
- String comparison is **case-insensitive**.
  - All the uppercase letters come before all the lowercase letters

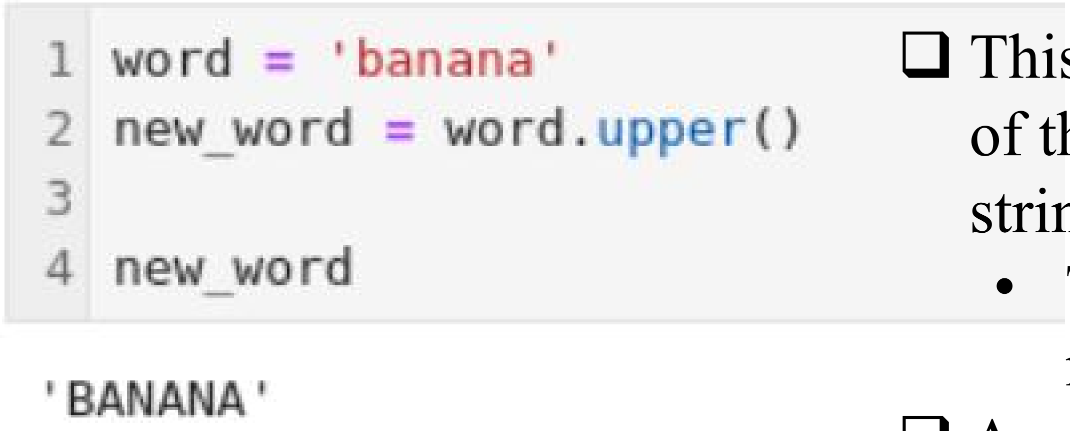


```
def compare_with_banana(word):
    if word < 'banana':
        print(word, 'comes before banana.')
    elif word > 'banana':
        print(word, 'comes after banana.')
    else:
        print('All right, banana.')

compare_with_banana('apple')
```

```
apple comes before banana.
```

5. String Methods

- Strings provide methods that perform a variety of useful operations.
- A method is similar to a function—it takes arguments and returns a value—but the syntax is different. For example, the method upper takes a string and returns a new string with all uppercase letters.



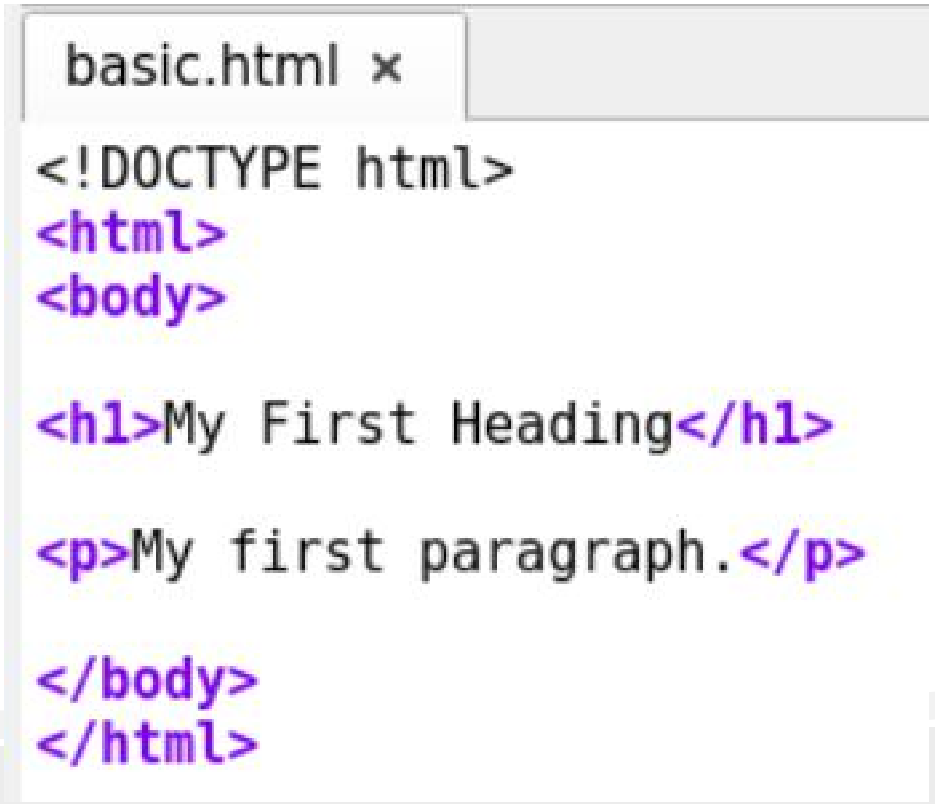
```
word = 'banana'
new_word = word.upper()

new_word
```

```
'BANANA'
```

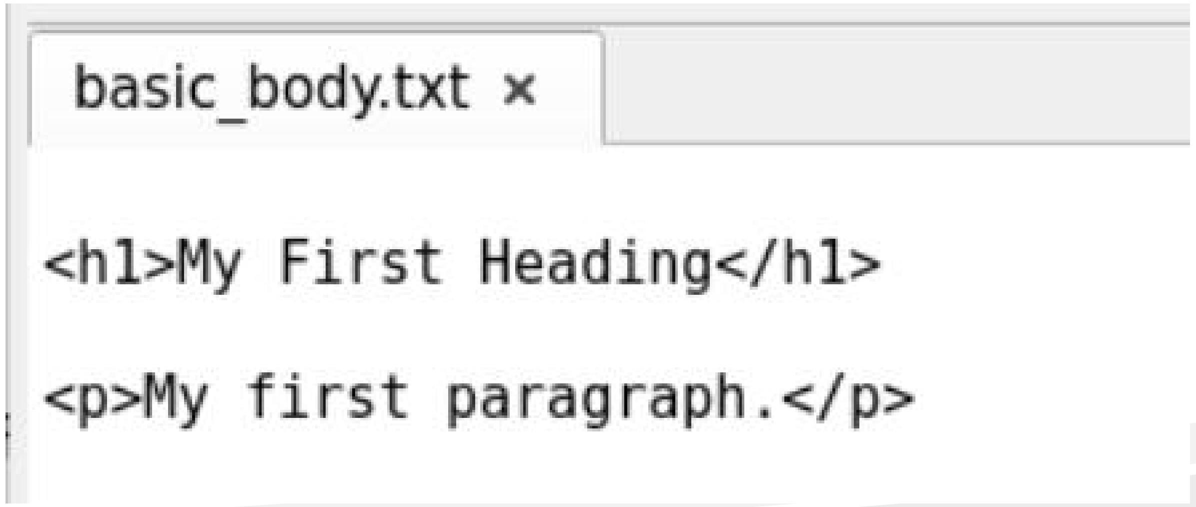
- ❑ This use of the **dot** operator specifies the name of the method, **upper**, and the name of the string to apply the method to, **word**.
  - The empty parentheses indicate that this method takes no arguments.
- ❑ A method call is called an **invocation**; in this case, we would say that we are invoking upper on word.

6. Writing Files

- Reading the file *basic.html* and writing the body of the html to a new file *basic_body.txt*



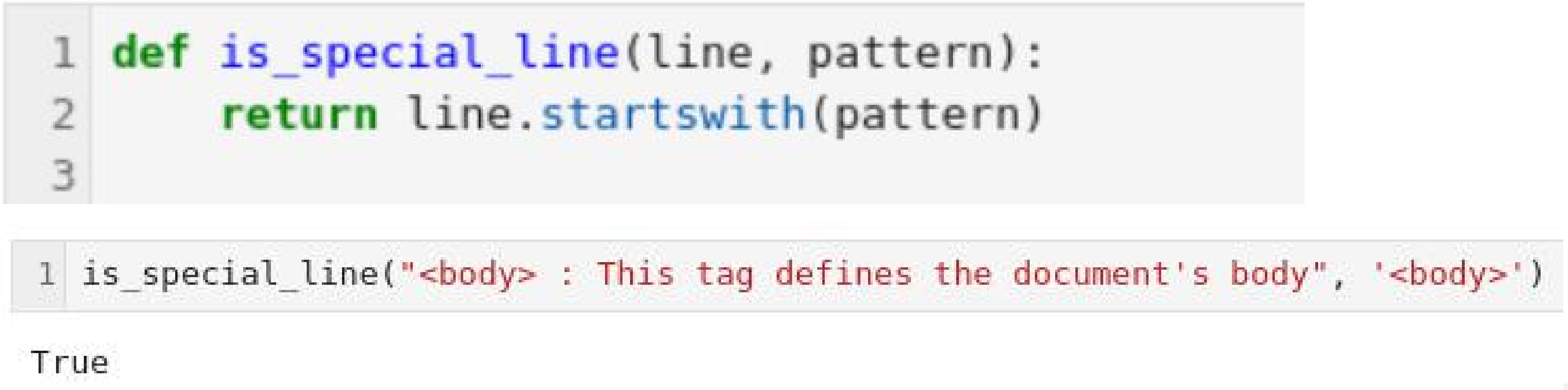
```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```



```

<h1>My First Heading</h1>

<p>My first paragraph.</p>
```

6. Writing Files

- The following function takes a line and checks whether it is one of the special lines. It uses the **startswith** method, which checks whether a string starts with a given sequence of characters:



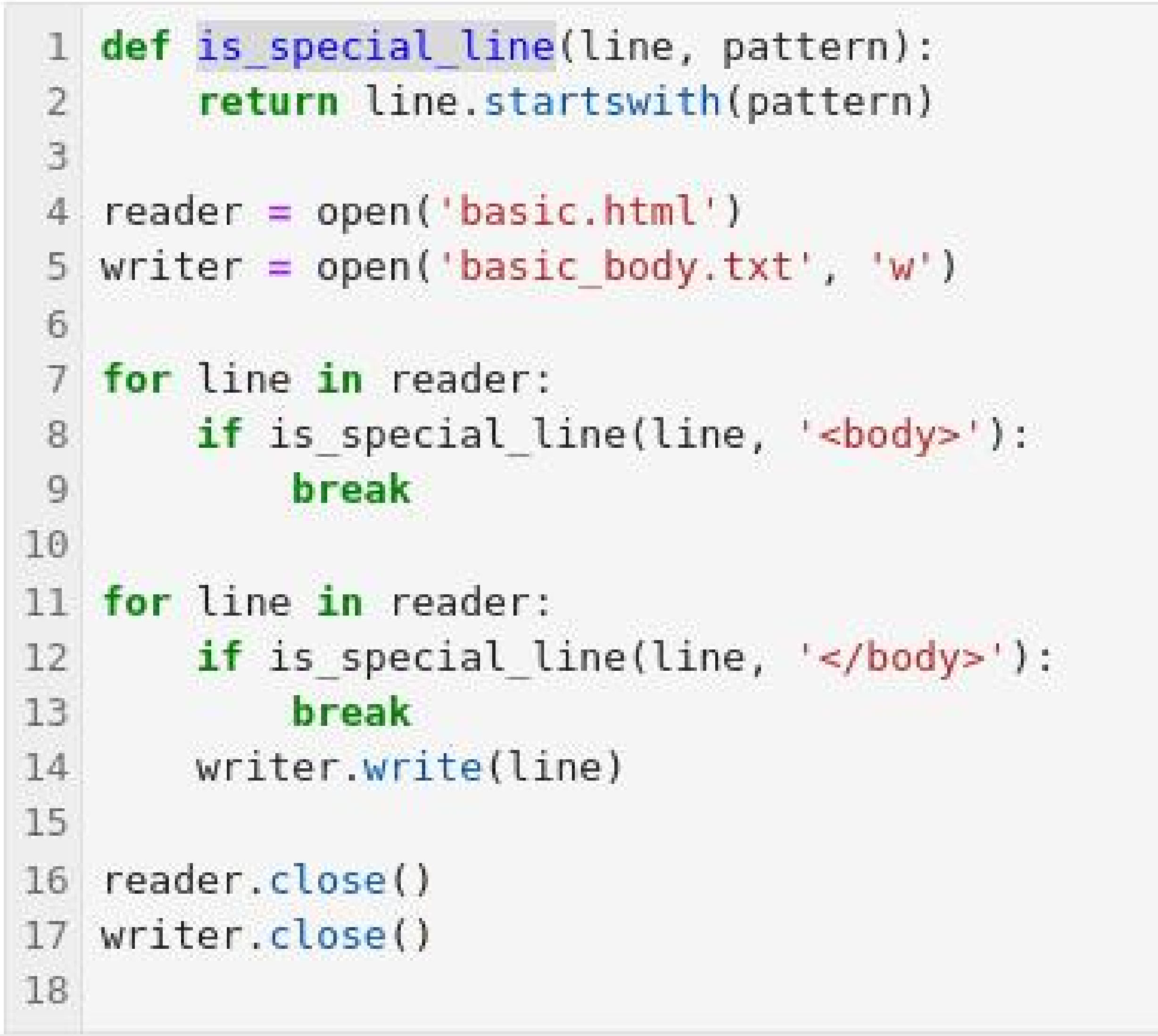
```python
def is_special_line(line, pattern):
    return line.startswith(pattern)

```

```python
is_special_line("<body> : This tag defines the document's body", '<body>')
```

```
True
```

6. Writing Files

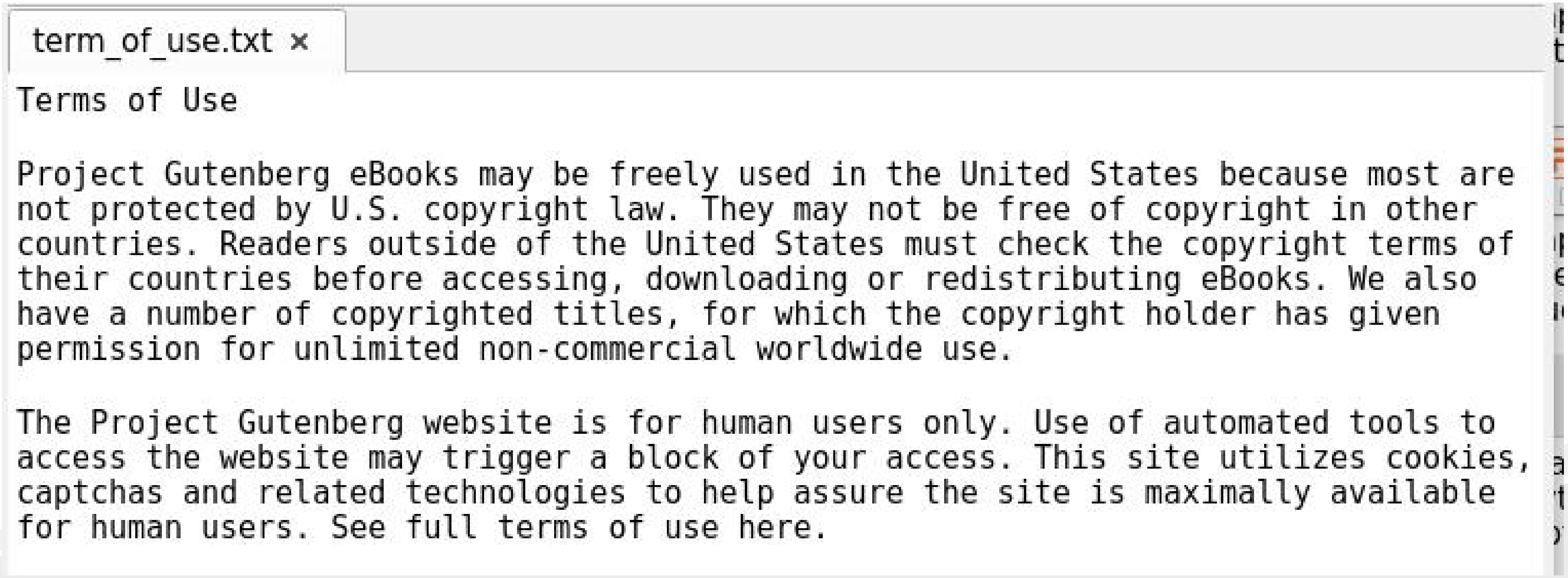
- In the first loop, we’ll read through the file *basic.html* until we find the first line which is started with “**<body>**” :
  - The **break** statement “breaks” out of the loop—that is, it causes the loop to end immediately, before we get to the end of the file.
- The second loop, we reads the rest of the file *basic.html*, one line at a time. When it finds the special line “**</body>**” that indicates the end of the body, it breaks out of the loop. Otherwise, it writes the line to the output file *basic_body.txt*



```python
def is_special_line(line, pattern):
    return line.startswith(pattern)

reader = open('basic.html')
writer = open('basic_body.txt', 'w')

for line in reader:
    if is_special_line(line, '<body>'):
        break

for line in reader:
    if is_special_line(line, '</body>'):
        break
    writer.write(line)

reader.close()
writer.close()
```

7. Find and Replace

- Find and count the number of times the word '***copyright***' appears in the file *term_of_use.txt*.

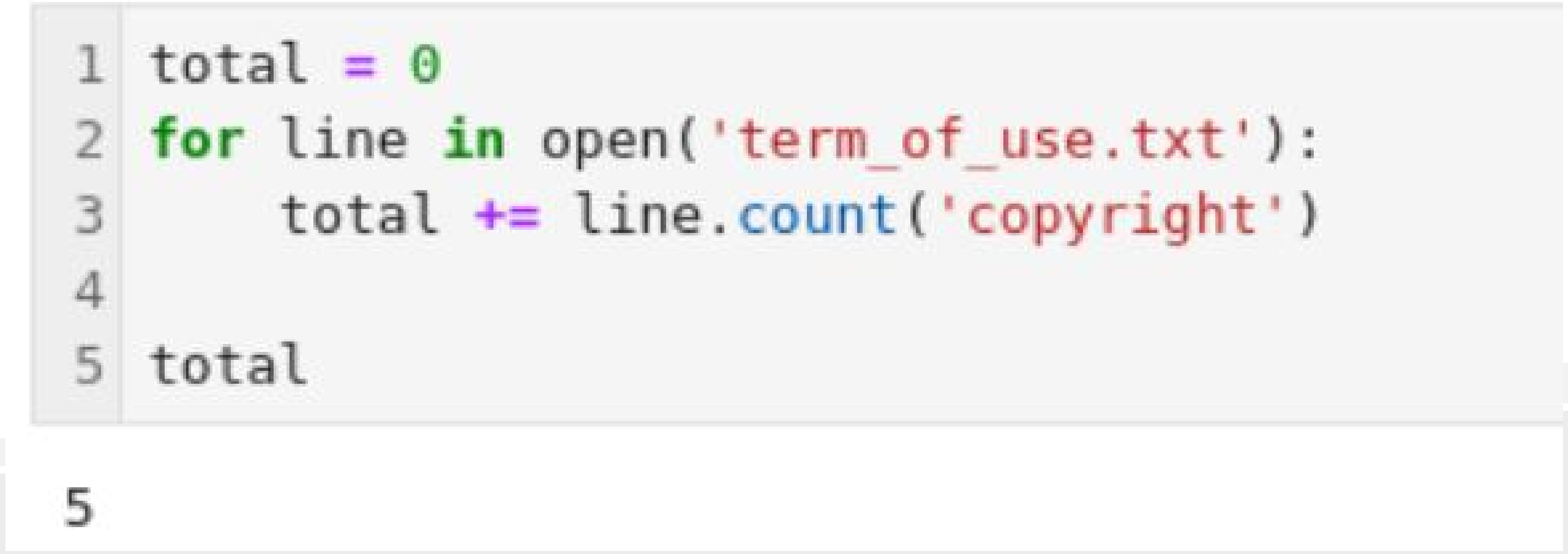


term_of_use.txt ×

```
Terms of Use

Project Gutenberg eBooks may be freely used in the United States because most are
not protected by U.S. copyright law. They may not be free of copyright in other
countries. Readers outside of the United States must check the copyright terms of
their countries before accessing, downloading or redistributing eBooks. We also
have a number of copyrighted titles, for which the copyright holder has given
permission for unlimited non-commercial worldwide use.

The Project Gutenberg website is for human users only. Use of automated tools to
access the website may trigger a block of your access. This site utilizes cookies,
captchas and related technologies to help assure the site is maximally available
for human users. See full terms of use here.
```

7. Find and Replace

- The **count** method of a string
  - Returns the number of times a sequence appears in a string
- Find and count the number of times the word '***copyright***' appears in the file *term_of_use.txt*.

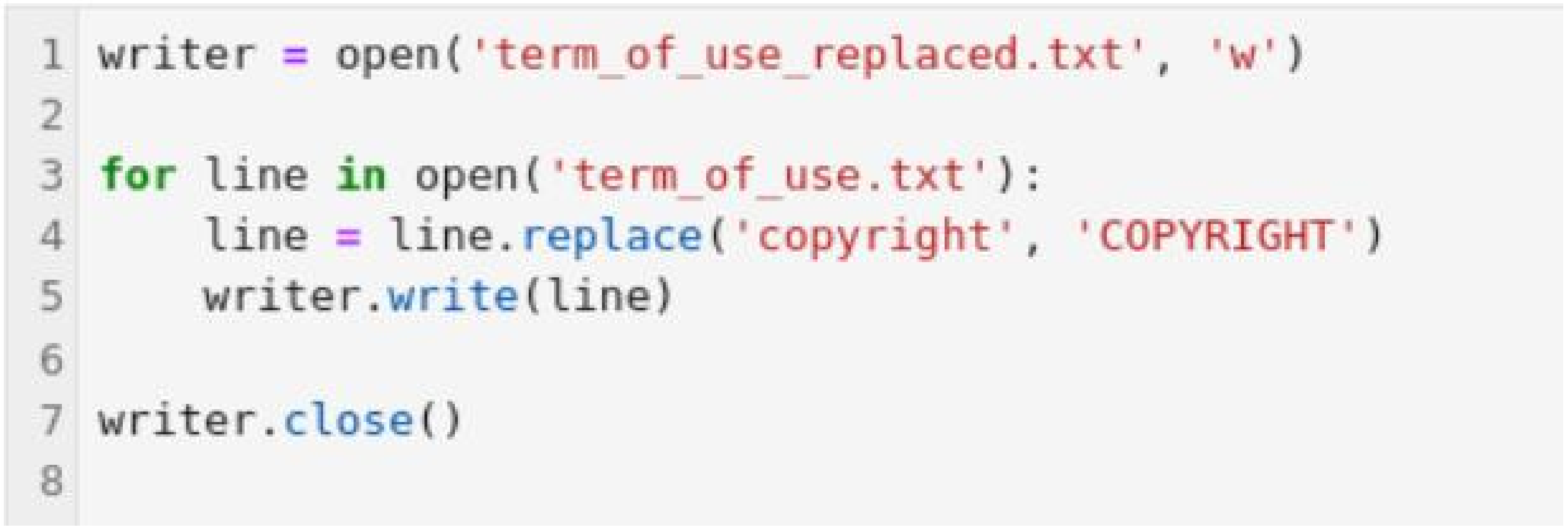


```python
total = 0
for line in open('term_of_use.txt'):
    total += line.count('copyright')

total
```

```
5
```

7. Find and Replace

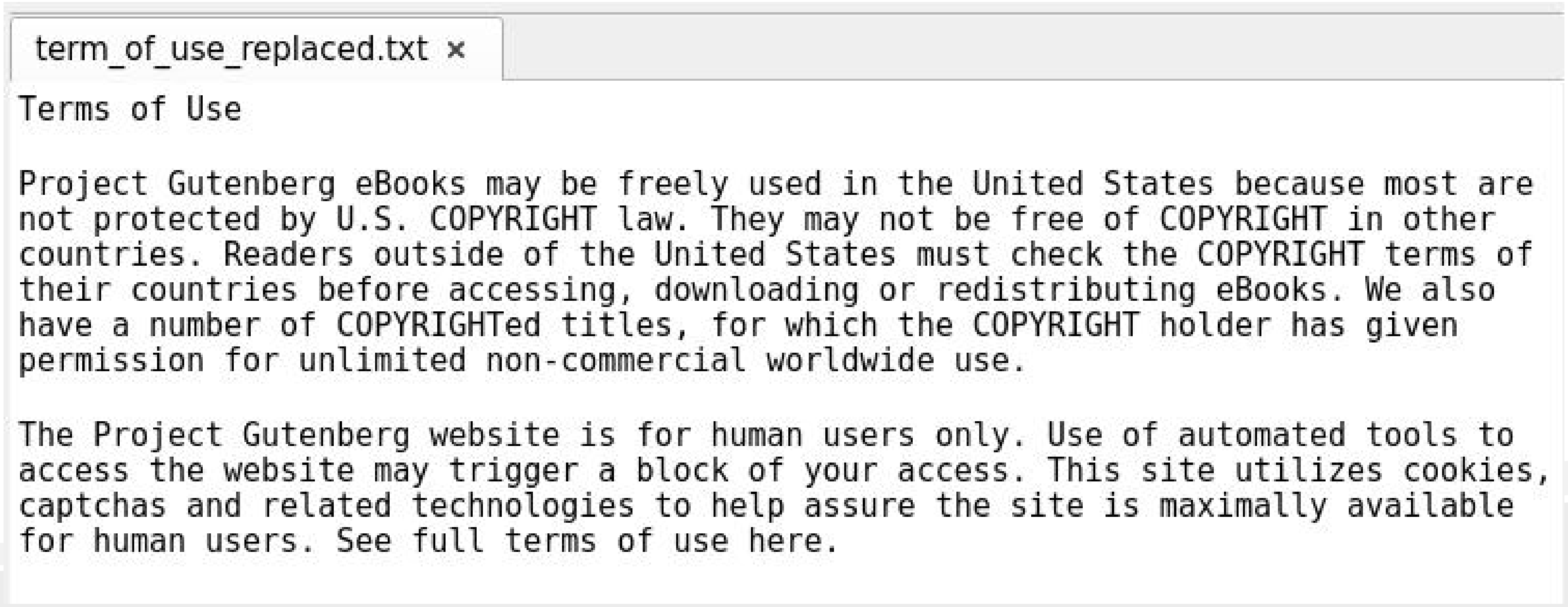
- Replace ‘**copyright**’ with ‘**COPYRIGHT**’ and write to a new file *term_of_use_replaced.txt*



```
writer = open('term_of_use_replaced.txt', 'w')

for line in open('term_of_use.txt'):
    line = line.replace('copyright', 'COPYRIGHT')
    writer.write(line)

writer.close()

```

7. Find and Replace

- The result is a new file called *term_of_use_replaced.txt* where ‘copyright’ is replaced by ‘COPYRIGHT’.


term_of_use_replaced.txt ×

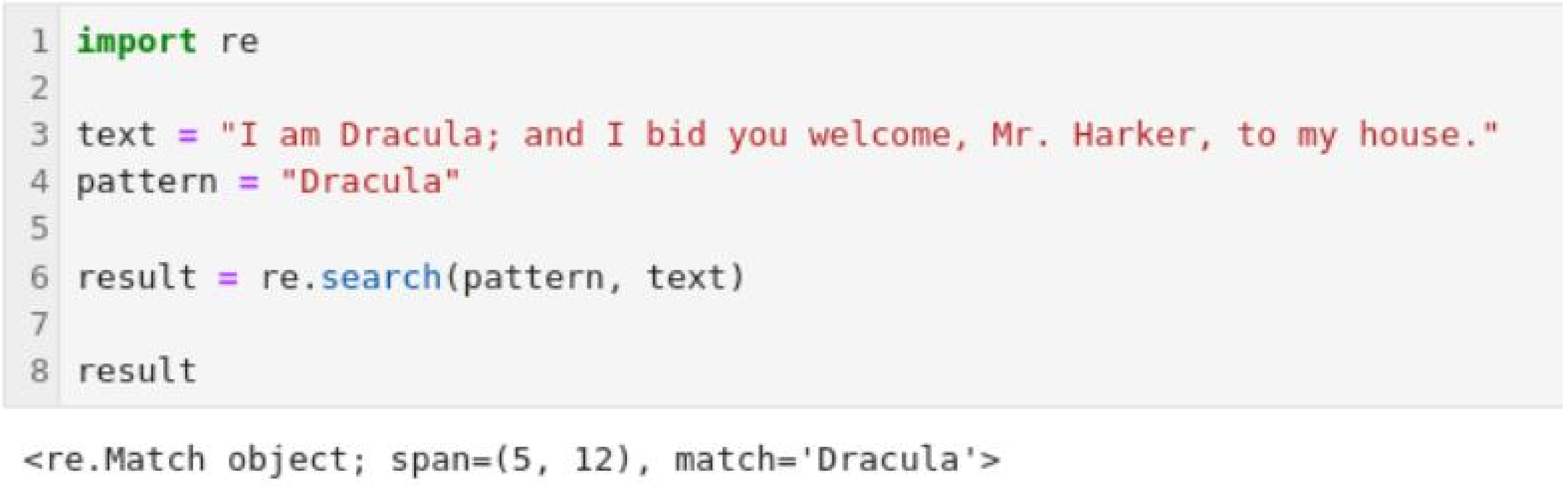
```
Terms of Use

Project Gutenberg eBooks may be freely used in the United States because most are
not protected by U.S. COPYRIGHT law. They may not be free of COPYRIGHT in other
countries. Readers outside of the United States must check the COPYRIGHT terms of
their countries before accessing, downloading or redistributing eBooks. We also
have a number of COPYRIGHTed titles, for which the COPYRIGHT holder has given
permission for unlimited non-commercial worldwide use.

The Project Gutenberg website is for human users only. Use of automated tools to
access the website may trigger a block of your access. This site utilizes cookies,
captchas and related technologies to help assure the site is maximally available
for human users. See full terms of use here.
```

8. Regular Expressions

- If we know exactly what sequence of characters we’re looking for, we can use the **in** operator to find it and the **replace** method to replace it. But there is another tool, called a **regular expression**, that can also perform these operations—and a lot more.
- Example: Find Dracula” in the text "I am Dracula; and I bid you welcome, Mr. Harker, to my house."
- A module called **re** provides functions related to regular expressions.
  - Use the **search** function to check whether the pattern appears in the text

8. Regular Expressions

- Use the **search** function of the **re** module:



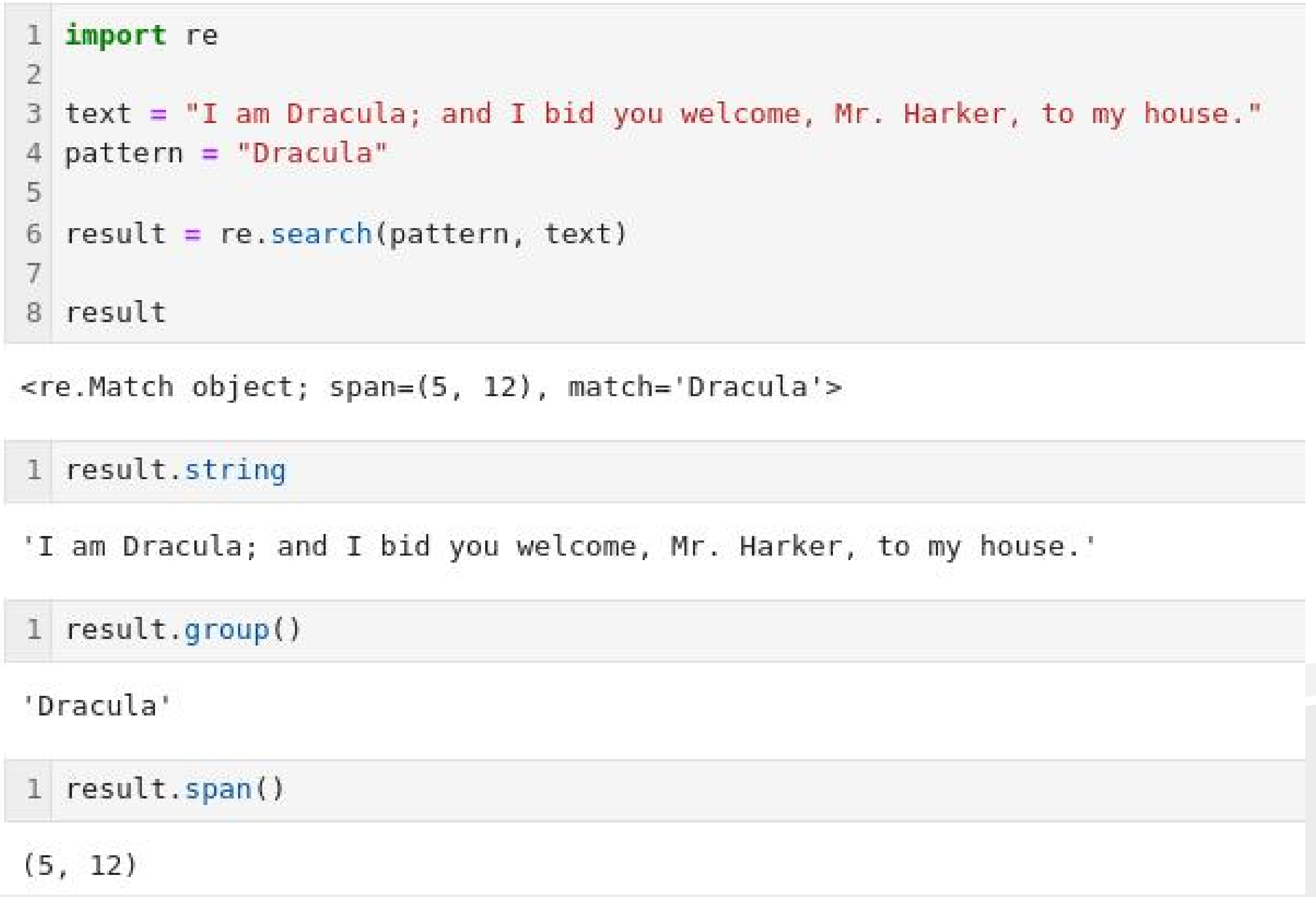
```python
import re

text = "I am Dracula; and I bid you welcome, Mr. Harker, to my house."
pattern = "Dracula"

result = re.search(pattern, text)

result
```

```
<re.Match object; span=(5, 12), match='Dracula'>
```

  – If the pattern appears in the text, **search** returns a **Match** object that contains the results of the search. Otherwise, the **search** function returns **None**.

8. Regular Expressions

- Match object that contains the results of the search.
  - **string** member: Contains the text that was searched.
  - **group()** member: Returns the part of the text that matched the pattern.
  - **span()** member: Returns the index in the text where the pattern starts and ends.



```python
import re

text = "I am Dracula; and I bid you welcome, Mr. Harker, to my house."
pattern = "Dracula"

result = re.search(pattern, text)

result
```

```
<re.Match object; span=(5, 12), match='Dracula'>
```

```python
result.string
```

```
'I am Dracula; and I bid you welcome, Mr. Harker, to my house.'
```

```python
result.group()
```

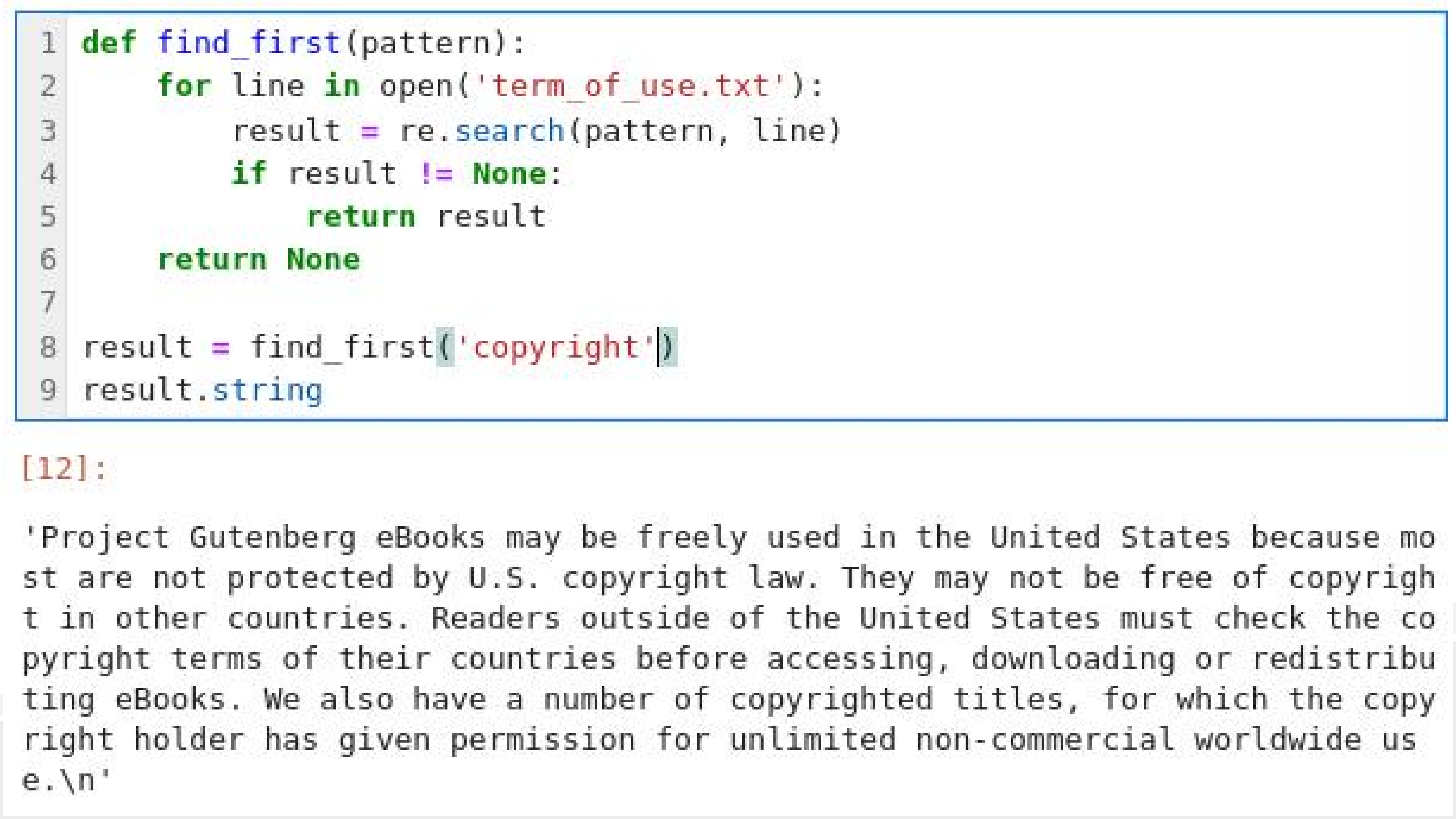
```
'Dracula'
```

```python
result.span()
```

```
(5, 12)
```

8. Regular Expressions

- Here's a function that loops through the lines in the file until it finds the first one that matches the given pattern, and returns the **Match** object:

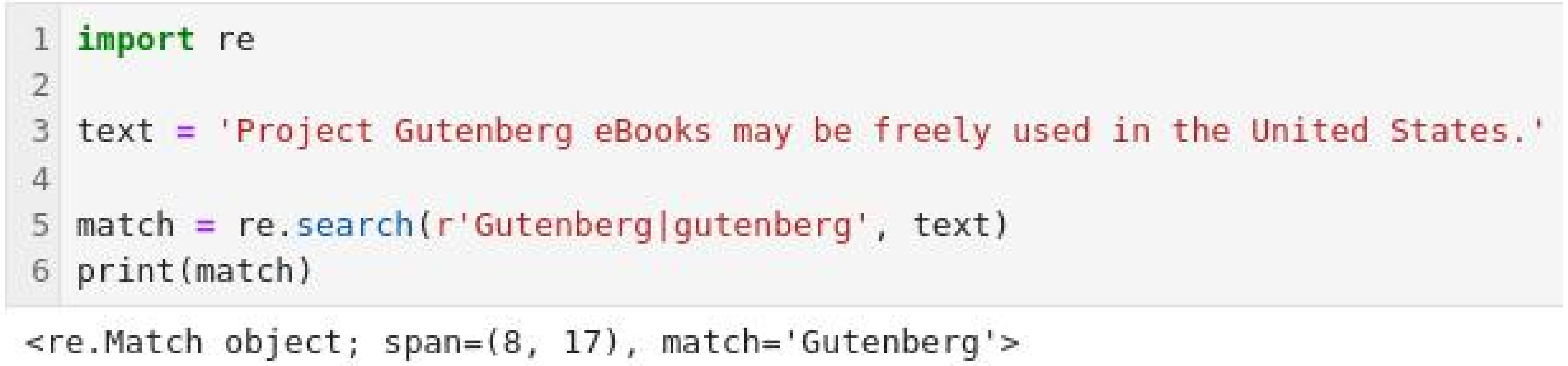


```
def find_first(pattern):
    for line in open('term_of_use.txt'):
        result = re.search(pattern, line)
        if result != None:
            return result
    return None

result = find_first('copyright')
result.string
```

```
[12]:

'Project Gutenberg eBooks may be freely used in the United States because mo
st are not protected by U.S. copyright law. They may not be free of copyrigh
t in other countries. Readers outside of the United States must check the co
pyright terms of their countries before accessing, downloading or redistribu
ting eBooks. We also have a number of copyrighted titles, for which the copy
right holder has given permission for unlimited non-commercial worldwide us
e.\n'
```

8. Regular Expressions

- https://docs.python.org/3/library/re.html#regular-expression-syntax
- If the pattern includes the vertical bar character, '|', it can match either the sequence on the left or the sequence on the right.

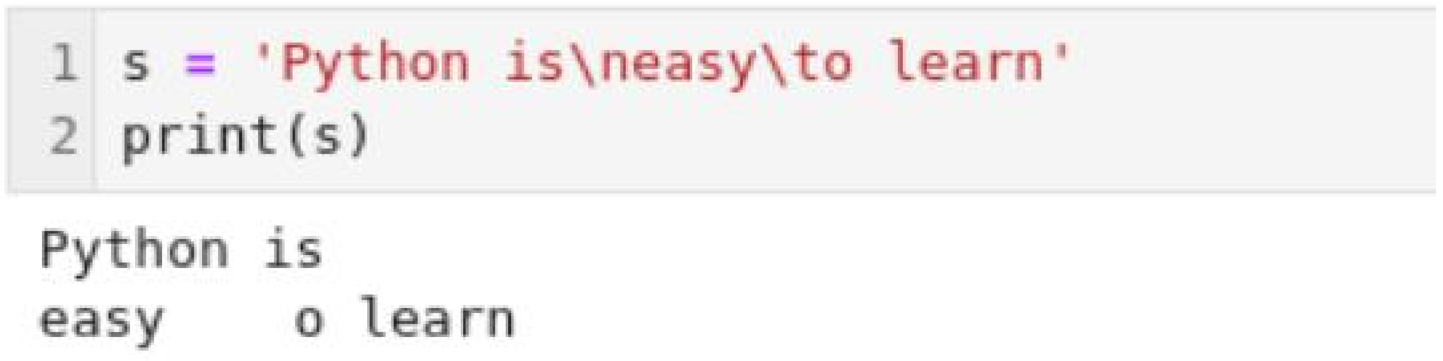


```python
import re

text = 'Project Gutenberg eBooks may be freely used in the United States.'

match = re.search(r'Gutenberg|gutenberg', text)
print(match)
```

```
<re.Match object; span=(8, 17), match='Gutenberg'>
```

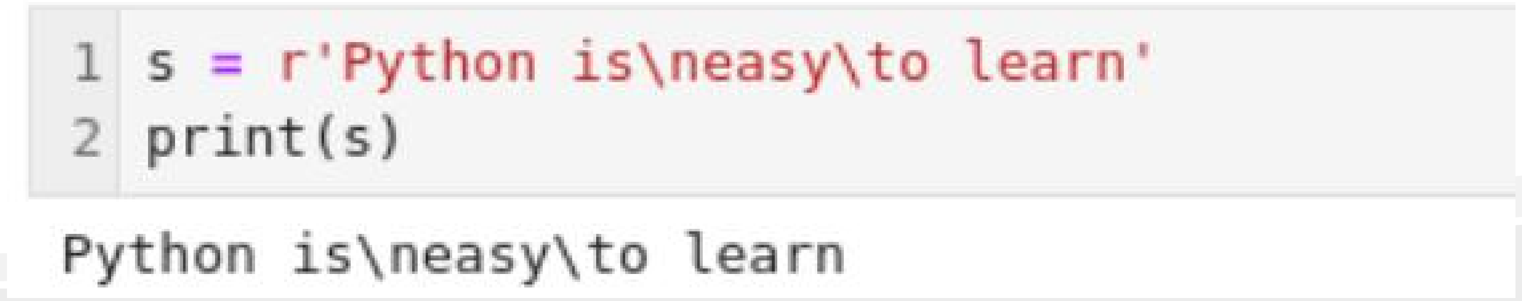
8. Regular Expressions

- In Python, a raw string is a special type of string that allows you to include backslashes (\) without interpreting them as escape sequences.



```python
s = 'Python is\neasy\to learn'
print(s)
```

```
Python is
easy	o learn
```

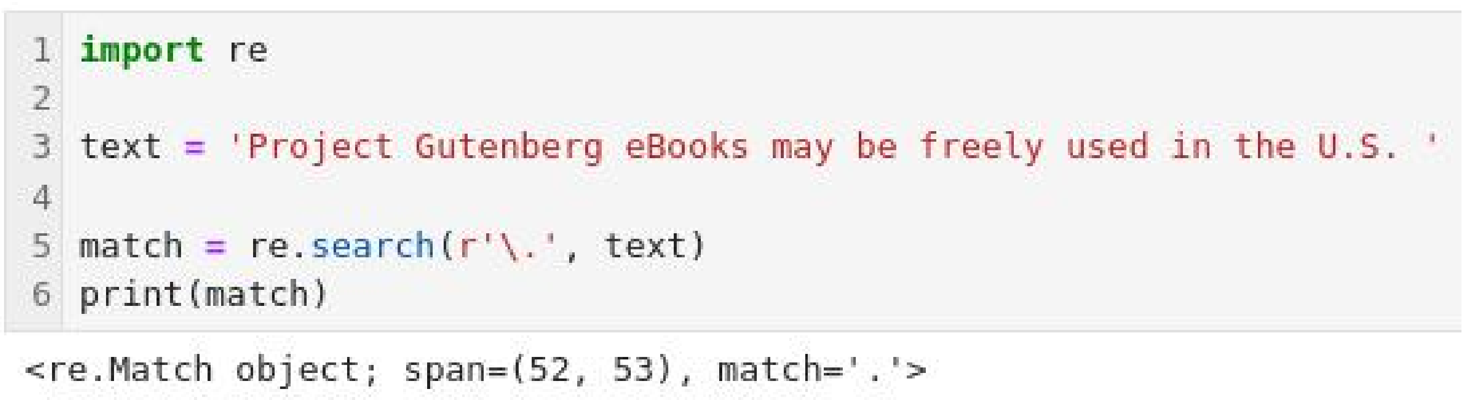


```python
s = r'Python is\neasy\to learn'
print(s)
```

```
Python is\neasy\to learn
```

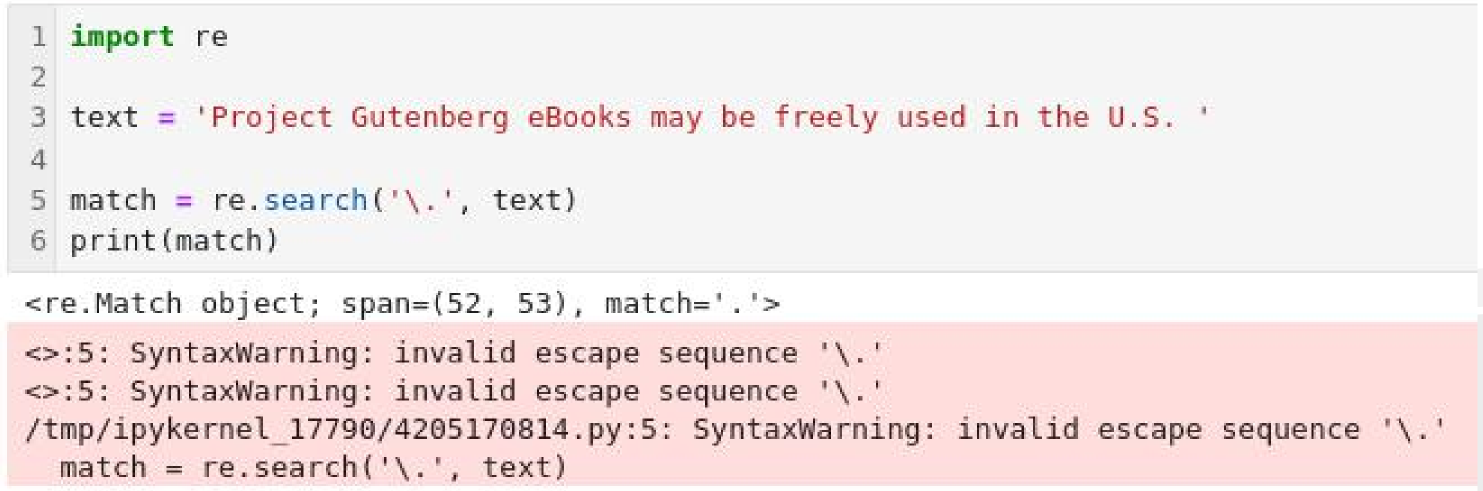
8. Regular Expressions

- Find the dot character in the text:



```python
import re

text = 'Project Gutenberg eBooks may be freely used in the U.S. '

match = re.search(r'\.', text)
print(match)
```

```
<re.Match object; span=(52, 53), match='.'>
```

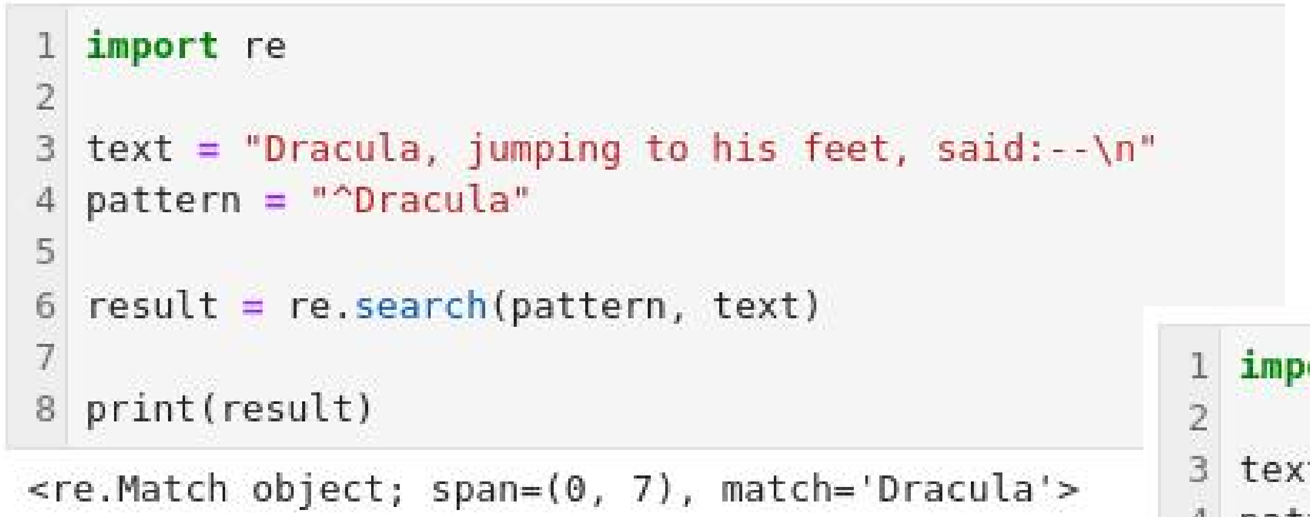


```python
import re

text = 'Project Gutenberg eBooks may be freely used in the U.S. '

match = re.search('\.', text)
print(match)
```

```
<re.Match object; span=(52, 53), match='.'>
<>:5: SyntaxWarning: invalid escape sequence '\.'
<>:5: SyntaxWarning: invalid escape sequence '\.'
/tmp/ipykernel_17790/4205170814.py:5: SyntaxWarning: invalid escape sequence '\.'
  match = re.search('\.', text)
```

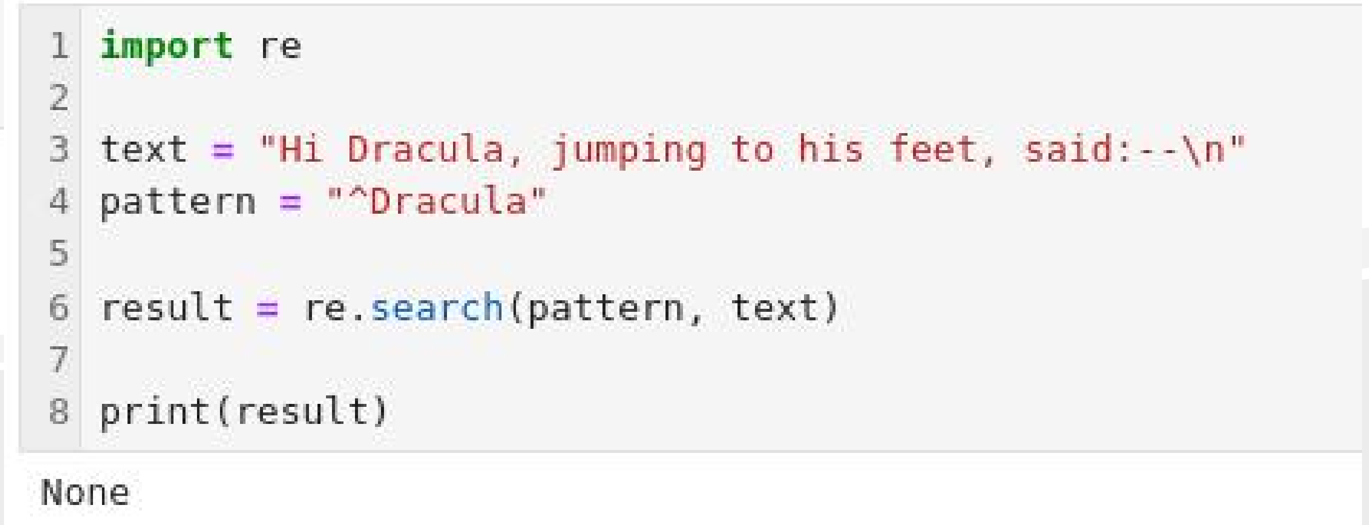
8. Regular Expressions

- The special character '^' matches the beginning of a string, so we can find a line that starts with a given pattern:



```python
import re

text = "Dracula, jumping to his feet, said:--\n"
pattern = "^Dracula"

result = re.search(pattern, text)

print(result)
```

```
<re.Match object; span=(0, 7), match='Dracula'>
```

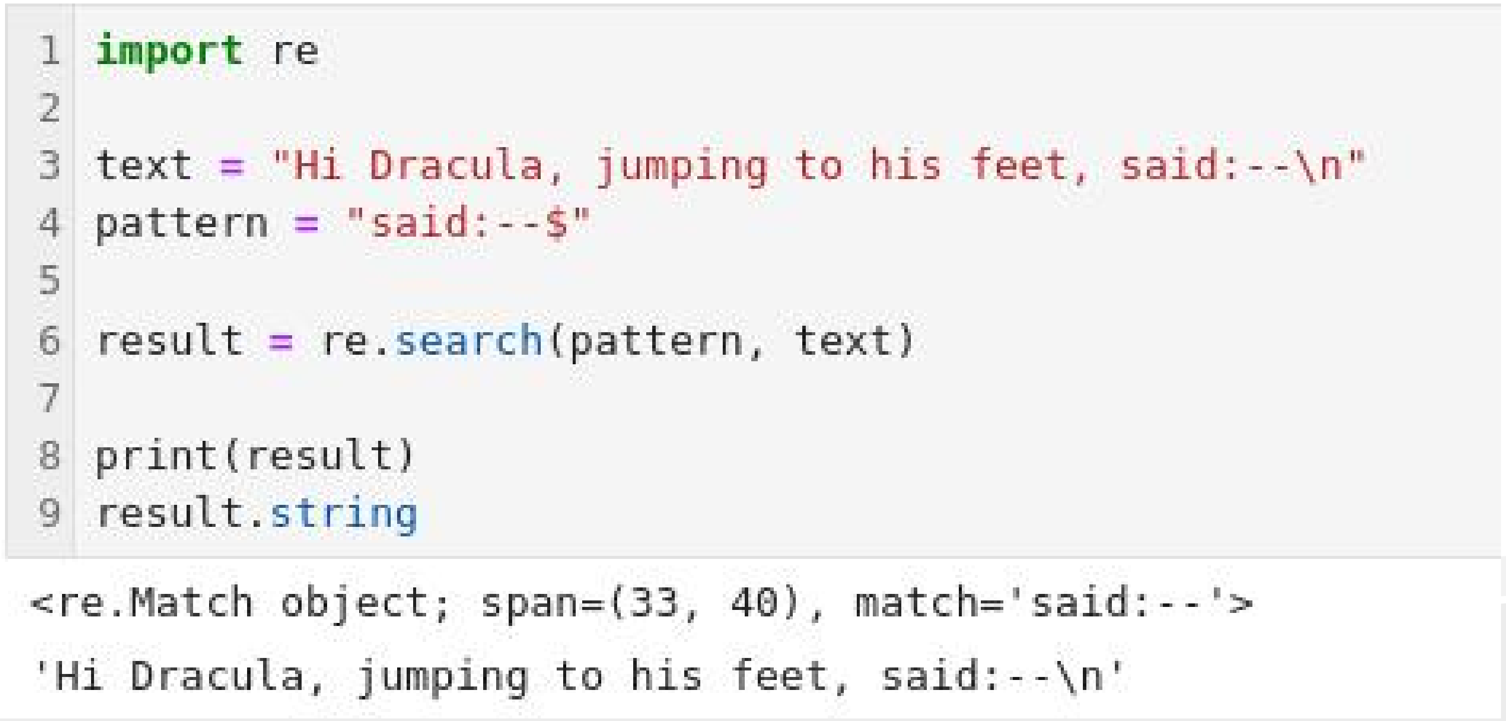


```python
import re

text = "Hi Dracula, jumping to his feet, said:--\n"
pattern = "^Dracula"

result = re.search(pattern, text)

print(result)
```

```
None
```

8. Regular Expressions

- And the special character '$' matches the end of a string, so we can find a line that ends with a given pattern (ignoring the newline at the end):

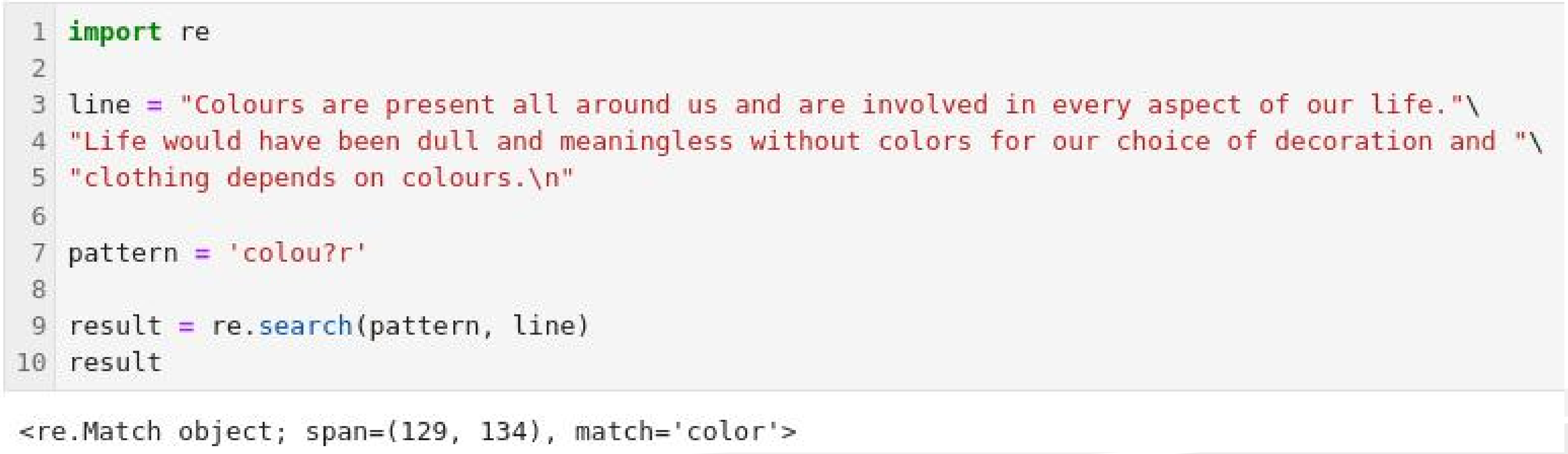


```python
import re

text = "Hi Dracula, jumping to his feet, said:--\n"
pattern = "said:--$"

result = re.search(pattern, text)

print(result)
result.string
```

```
<re.Match object; span=(33, 40), match='said:--'>
'Hi Dracula, jumping to his feet, said:--\n'
```

8. Regular Expressions

- The special character '?' in the pattern means that the previous character is optional:

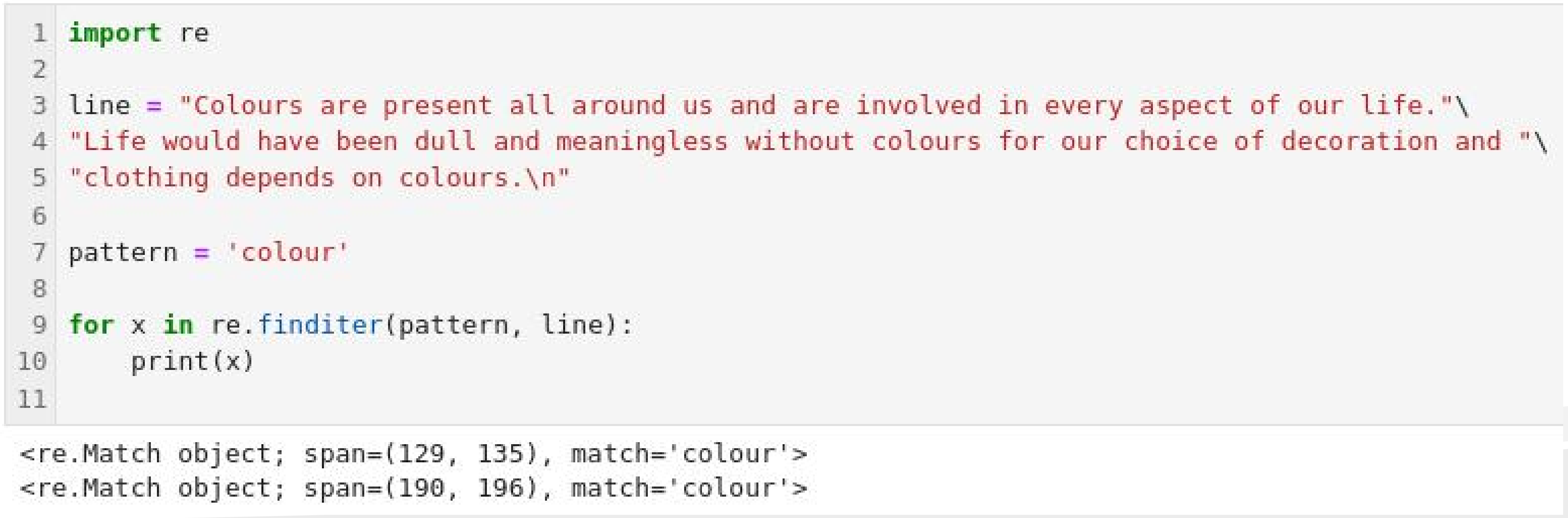


```
import re

line = "Colours are present all around us and are involved in every aspect of our life."\
"Life would have been dull and meaningless without colors for our choice of decoration and "\
"clothing depends on colours.\n"

pattern = 'colou?r'

result = re.search(pattern, line)
result
```

```
<re.Match object; span=(129, 134), match='color'>
```

8. Regular Expressions

- **re.finditer(pattern, text)**
  - Return an iterator yielding **Match** objects over all non-overlapping matches for the RE pattern in string.

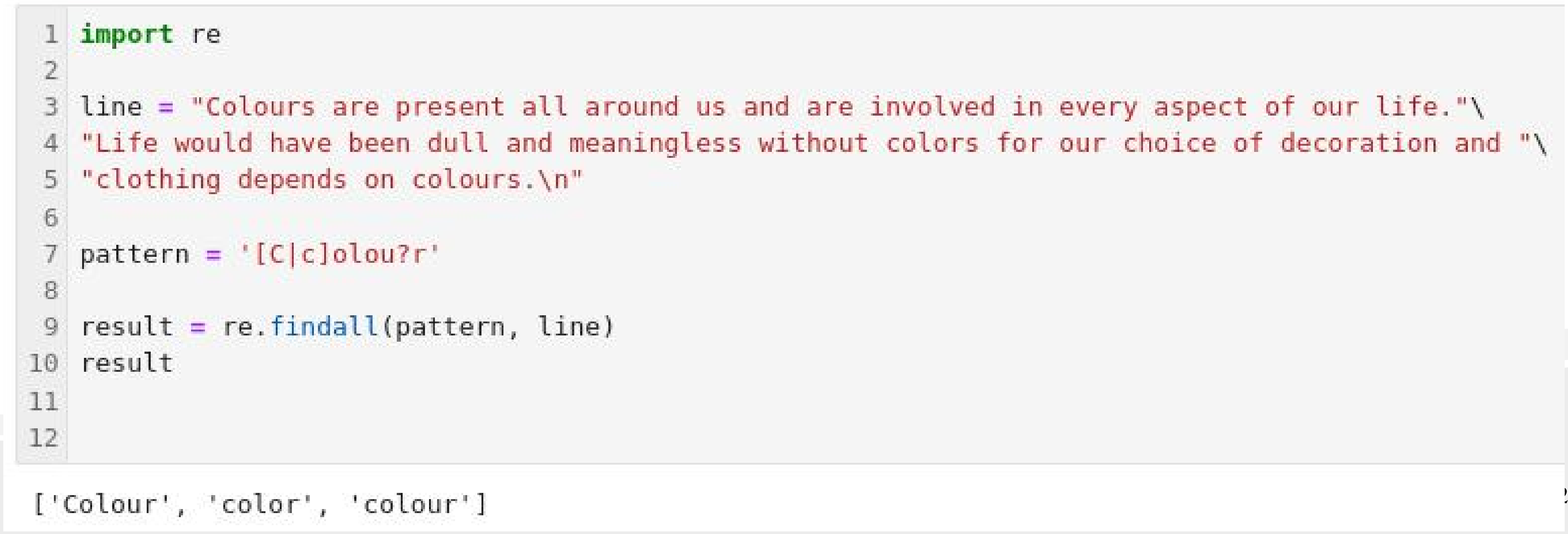


```python
import re

line = "Colours are present all around us and are involved in every aspect of our life."\
"Life would have been dull and meaningless without colours for our choice of decoration and "\
"clothing depends on colours.\n"

pattern = 'colour'

for x in re.finditer(pattern, line):
    print(x)

```

```
<re.Match object; span=(129, 135), match='colour'>
<re.Match object; span=(190, 196), match='colour'>
```

8. Regular Expressions

- **re.findall(pattern, text)**
  - Return all non-overlapping matches of pattern in string, as a list of strings or tuples. The string is scanned left-to-right, and matches are returned in the order found. Empty matches are included in the result.

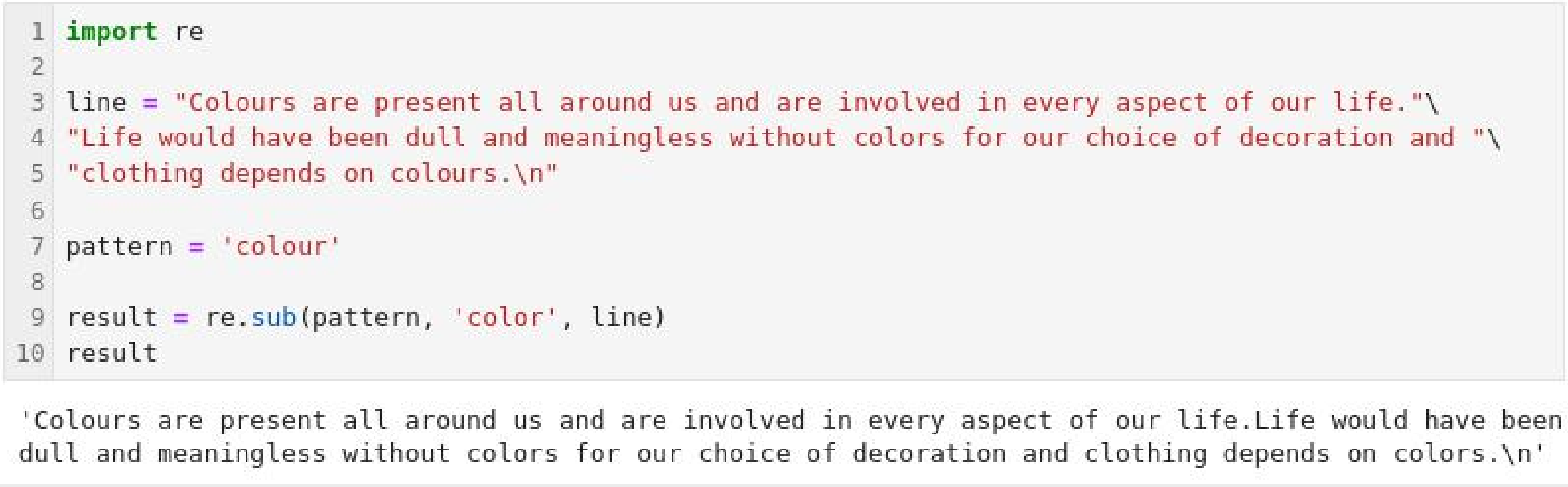


```python
import re

line = "Colours are present all around us and are involved in every aspect of our life."\
"Life would have been dull and meaningless without colors for our choice of decoration and "\
"clothing depends on colours.\n"

pattern = '[C|c]olou?r'

result = re.findall(pattern, line)
result


```

```
['Colour', 'color', 'colour']
```

9. String Substitution

- **sub** function in the **re** modul
  - The first argument is the pattern we want to find and replace, the second is what we want to replace it with, and the third is the string we want to search.
  - In the result, you can see that “colour” has been replaced with “color.”



```python
import re

line = "Colours are present all around us and are involved in every aspect of our life."\
"Life would have been dull and meaningless without colors for our choice of decoration and "\
"clothing depends on colours.\n"

pattern = 'colour'

result = re.sub(pattern, 'color', line)
result
```

```
'Colours are present all around us and are involved in every aspect of our life.Life would have been
dull and meaningless without colors for our choice of decoration and clothing depends on colors.\n'
```