# Python-Unit 4  -Part 1

Strings

## String in Python

A string is a sequence of characters.

A character is simply a symbol. For example, the English language has 26 characters.

Computers do not deal with characters, they deal with numbers (binary). Even though you may see characters on your screen, internally it is stored and manipulated as a combination of 0s and 1s.

This conversion of character to a number is called encoding, and the reverse process is decoding. ASCII and Unicode are some of the popular encodings used.

a string is a sequence of Unicode characters. Unicode was introduced to include every character in all languages and bring uniformity in encoding

## create a string in Python

Strings can be created by enclosing characters inside a single quote or double-quotes.
 Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

```
my_string = 'Hello'
print(my_string)
my_string = "Hello"
print(my_string)
my_string = '''Hello'''
print(my_string)
my_string = """Hello, welcome to
           the world of Python"""
print(my_string)
```

When you run the program, the output will be:

```
Hello
Hello
Hello
Hello, welcome to
the world of Python
```

## access characters in a string

We can access individual characters using indexing and a range of characters using slicing.

Index starts from 0.

Trying to access a character out of index range will raise an `IndexError`.

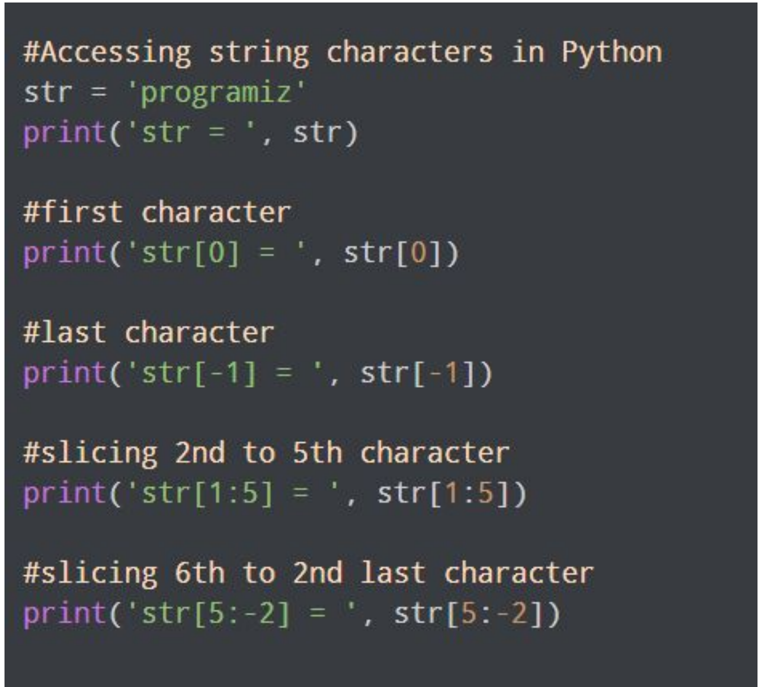
The index must be an integer.

can't use floats or other types, this will result into `TypeError`.

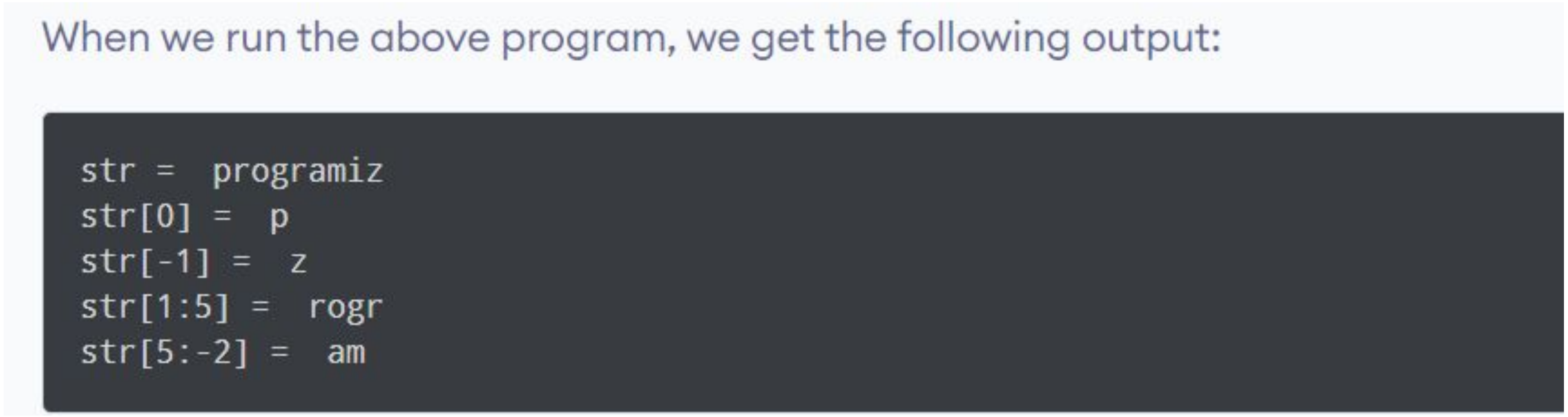
Python allows negative indexing for its sequences.

The index of `-1` refers to the last item, `-2` to the second last item and so on.

We can access a range of items in a string by using the slicing operator `:`(colon).



```python
#Accessing string characters in Python
str = 'programiz'
print('str = ', str)

#first character
print('str[0] = ', str[0])

#last character
print('str[-1] = ', str[-1])

#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])
```



When we run the above program, we get the following output:

```
str =  programiz
str[0] =  p
str[-1] =  z
str[1:5] =  rogr
str[5:-2] =  am
```

If we try to access an index out of the range or use numbers other than an integer, we will get errors.

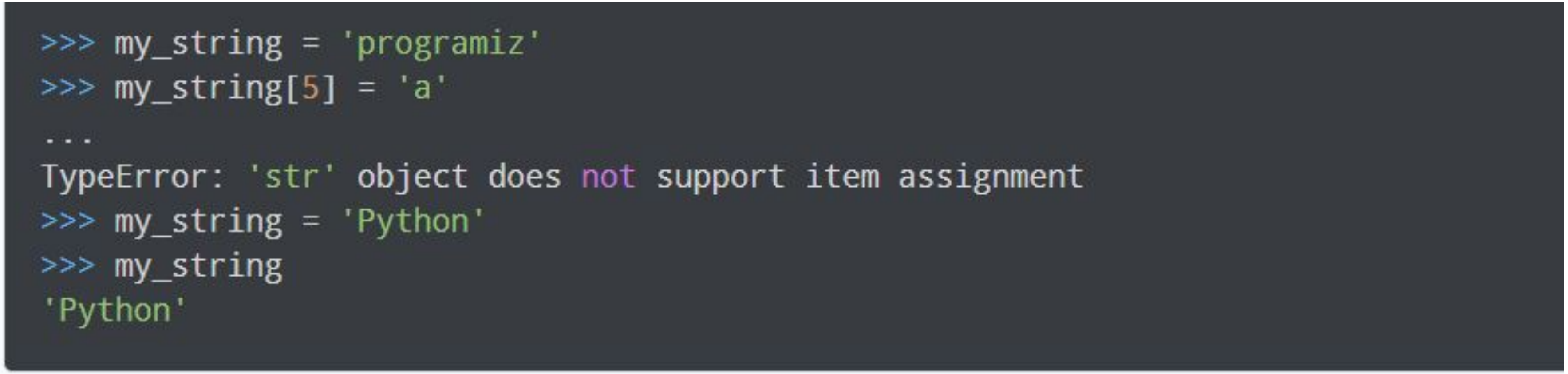
```
# index must be in range
>>> my_string[15]
...
IndexError: string index out of range

# index must be an integer
>>> my_string[1.5]
...
TypeError: string indices must be integers
```

## change or delete a string

Strings are immutable.

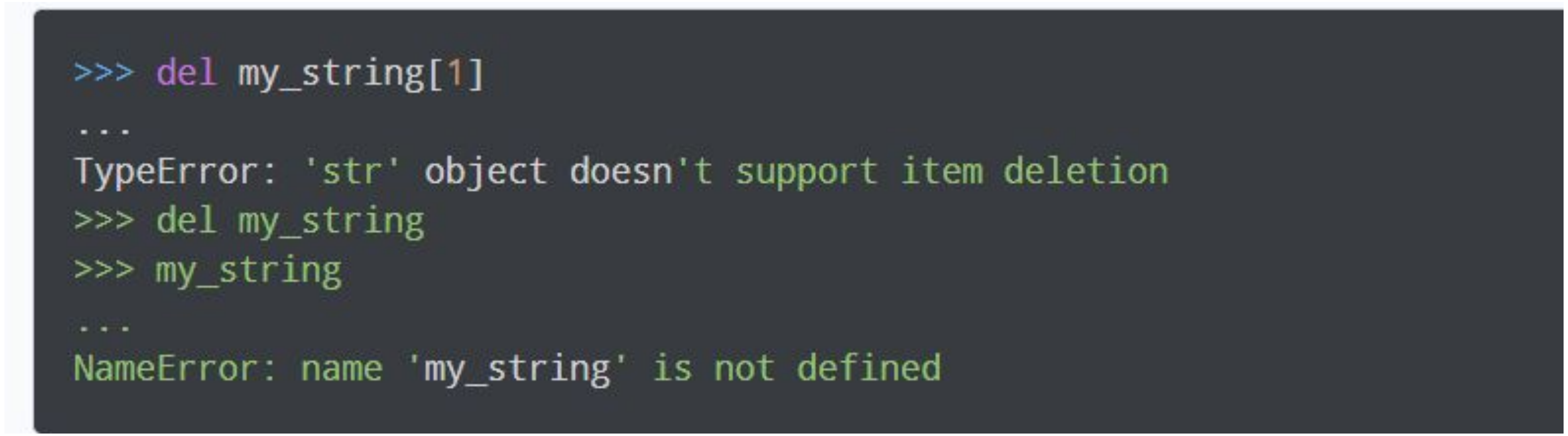
This means that elements of a string cannot be changed once they have been assigned.

We can simply reassign different strings to the same name.



```
>>> my_string = 'programiz'
>>> my_string[5] = 'a'
...
TypeError: 'str' object does not support item assignment
>>> my_string = 'Python'
>>> my_string
'Python'
```

We cannot delete or remove characters from a string. But deleting the string entirely is possible using the `del` keyword.


```
>>> del my_string[1]
...
TypeError: 'str' object doesn't support item deletion
>>> del my_string
>>> my_string
...
NameError: name 'my_string' is not defined
```

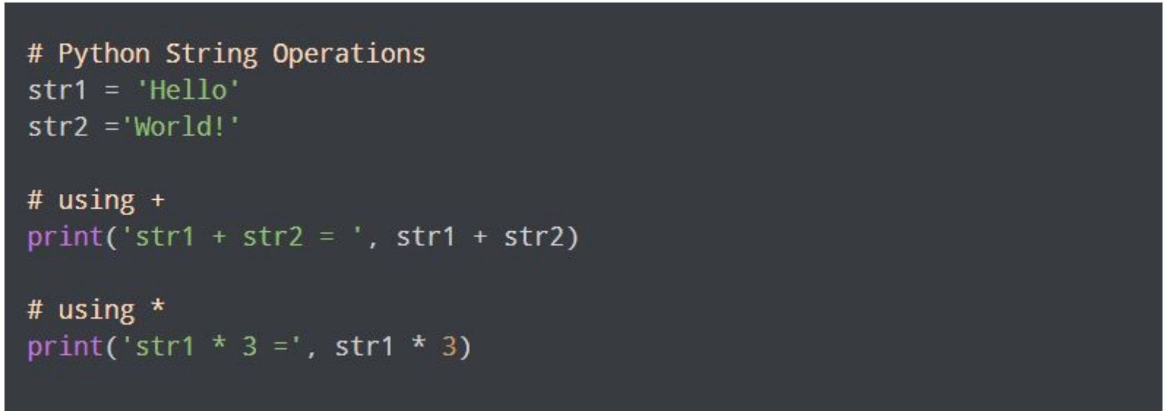
## Python String Operations

### Concatenation of Two or More Strings

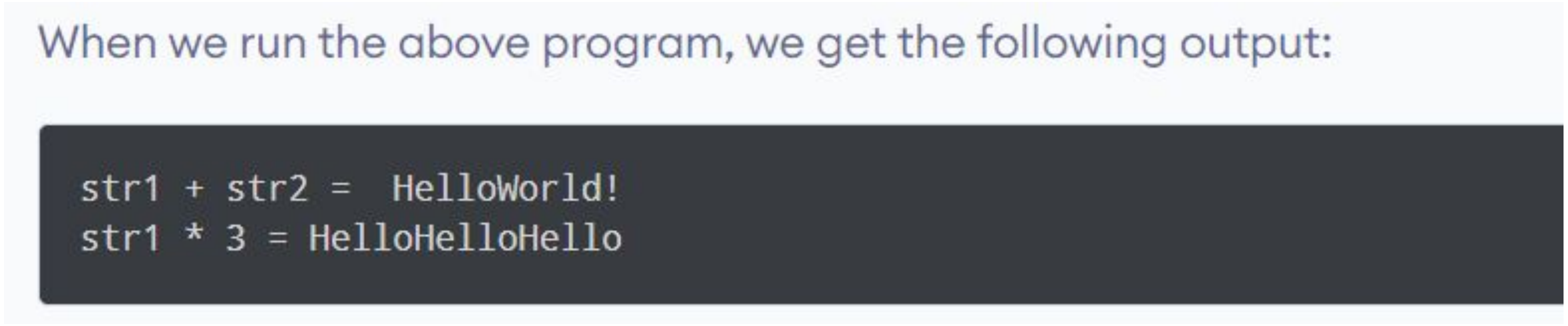
Joining of two or more strings into a single one is called concatenation.

The + operator does this in Python. Simply writing two string literals together also concatenates them.

The * operator can be used to repeat the string for a given number of times.



```python
# Python String Operations
str1 = 'Hello'
str2 ='World!'

# using +
print('str1 + str2 = ', str1 + str2)

# using *
print('str1 * 3 =', str1 * 3)
```

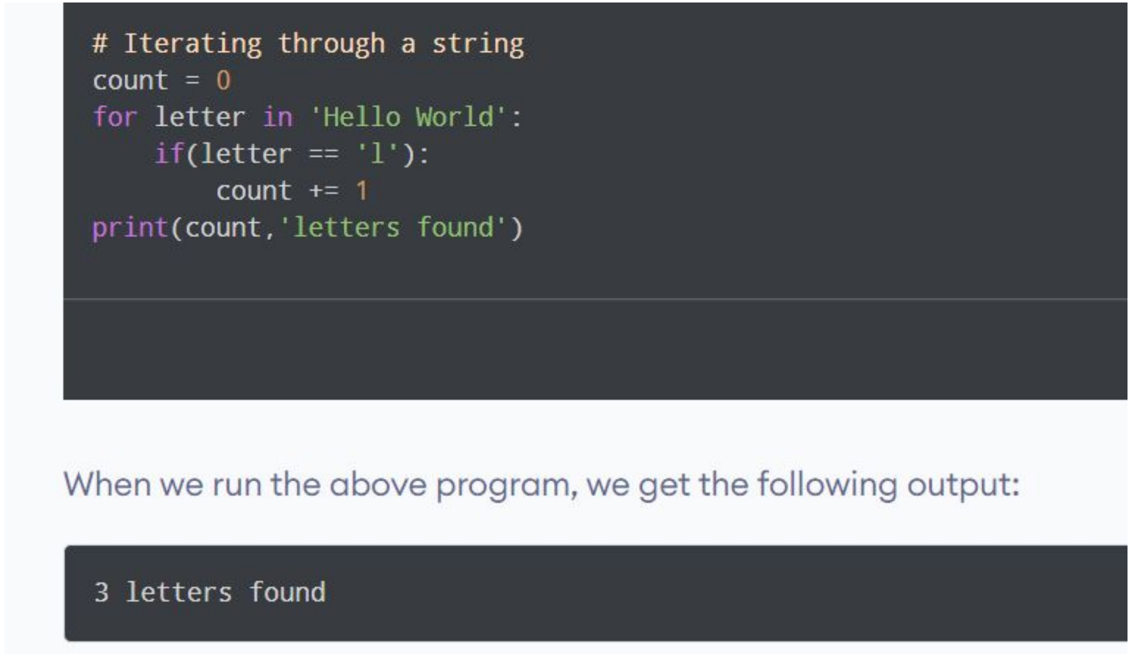


When we run the above program, we get the following output:

```
str1 + str2 =  HelloWorld!
str1 * 3 = HelloHelloHello
```

## Iterating Through a string

We can iterate through a string using a for loop. Here is an example to count the number of 'l's in a string.



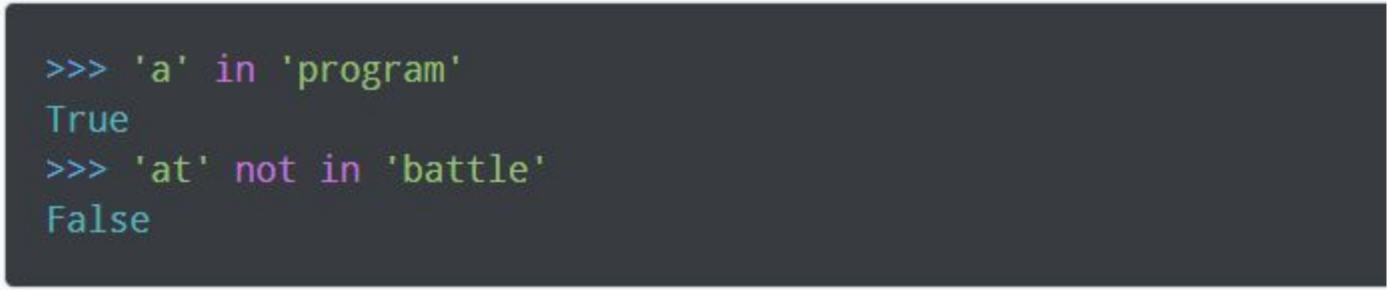
```python
# Iterating through a string
count = 0
for letter in 'Hello World':
    if(letter == 'l'):
        count += 1
print(count,'letters found')
```

When we run the above program, we get the following output:

```
3 letters found
```

## String Membership Test

We can test if a substring exists within a string or not, using the keyword `in`.



```
>>> 'a' in 'program'
True
>>> 'at' not in 'battle'
False
```

## Built-in functions to Work with Python

**enumerate()**

**len()**

**format()**

capitalize()

**lower()**

**upper()**

**join()**

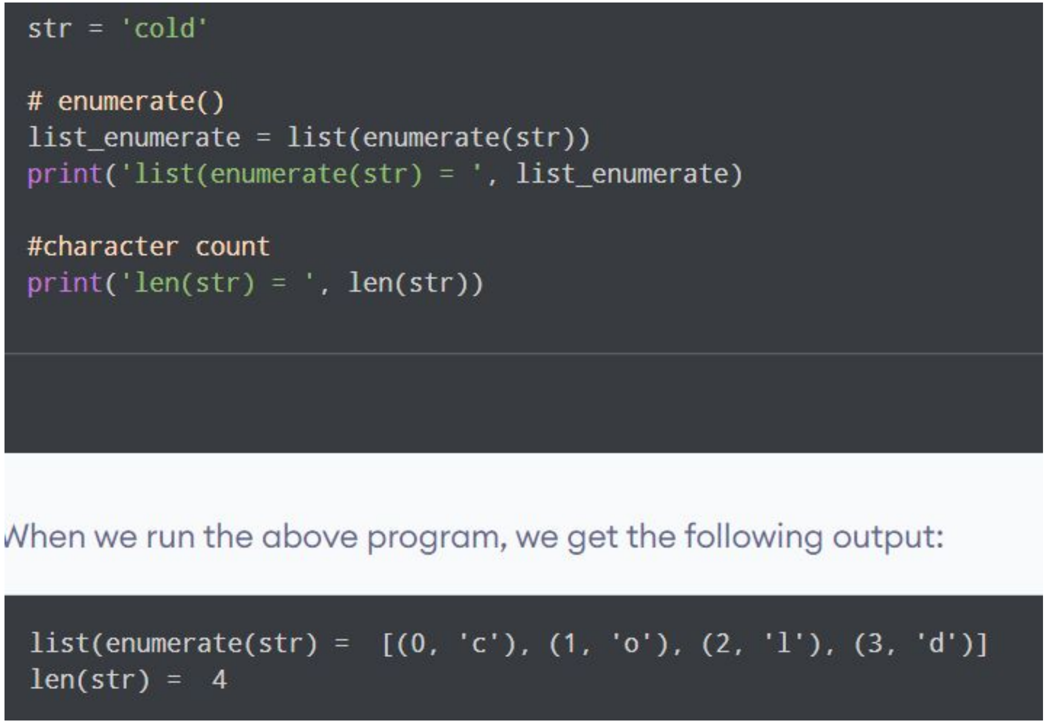
**split()**

**find()**

**replace()**

**`enumerate()`**-The `enumerate()` function returns an enumerate object. It contains the index and value of all the items in the string as pairs. This can be useful for iteration.

**`len()`**-`len()` returns the length (nu

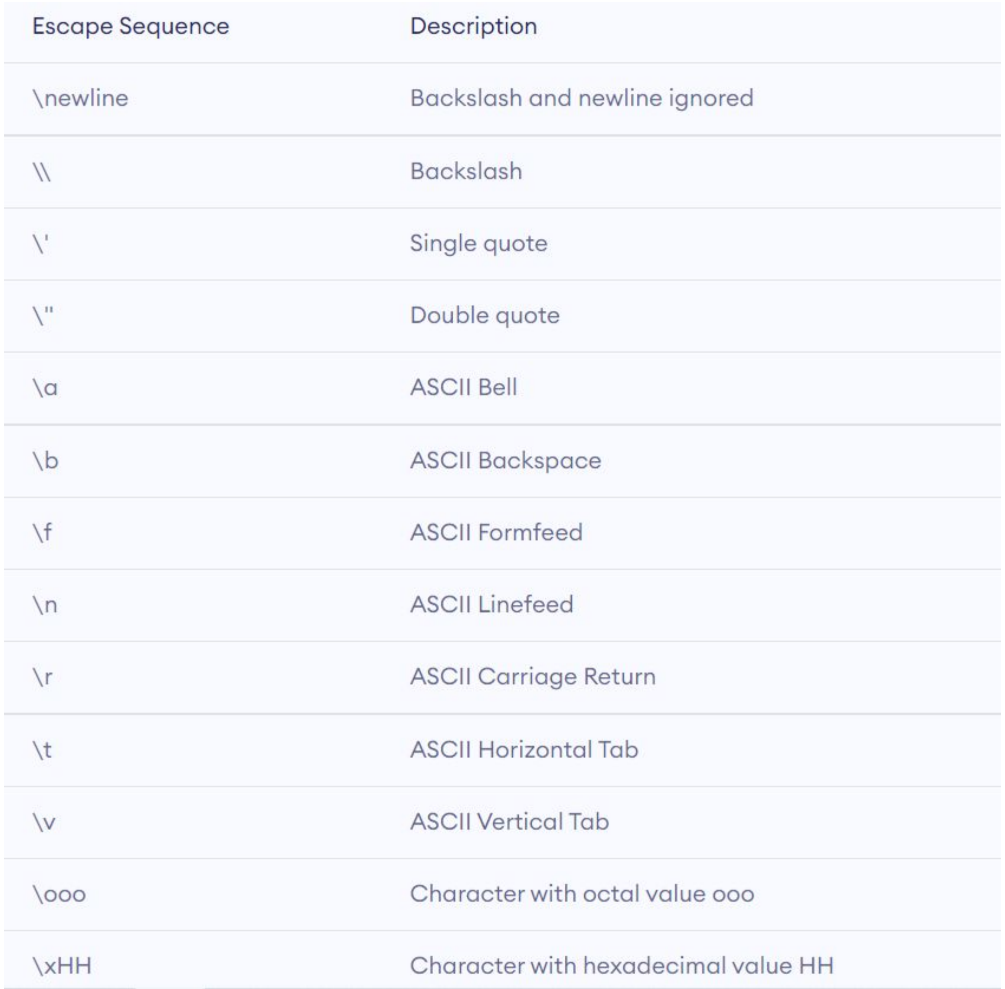


```
str = 'cold'

# enumerate()
list_enumerate = list(enumerate(str))
print('list(enumerate(str) = ', list_enumerate)

#character count
print('len(str) = ', len(str))
```

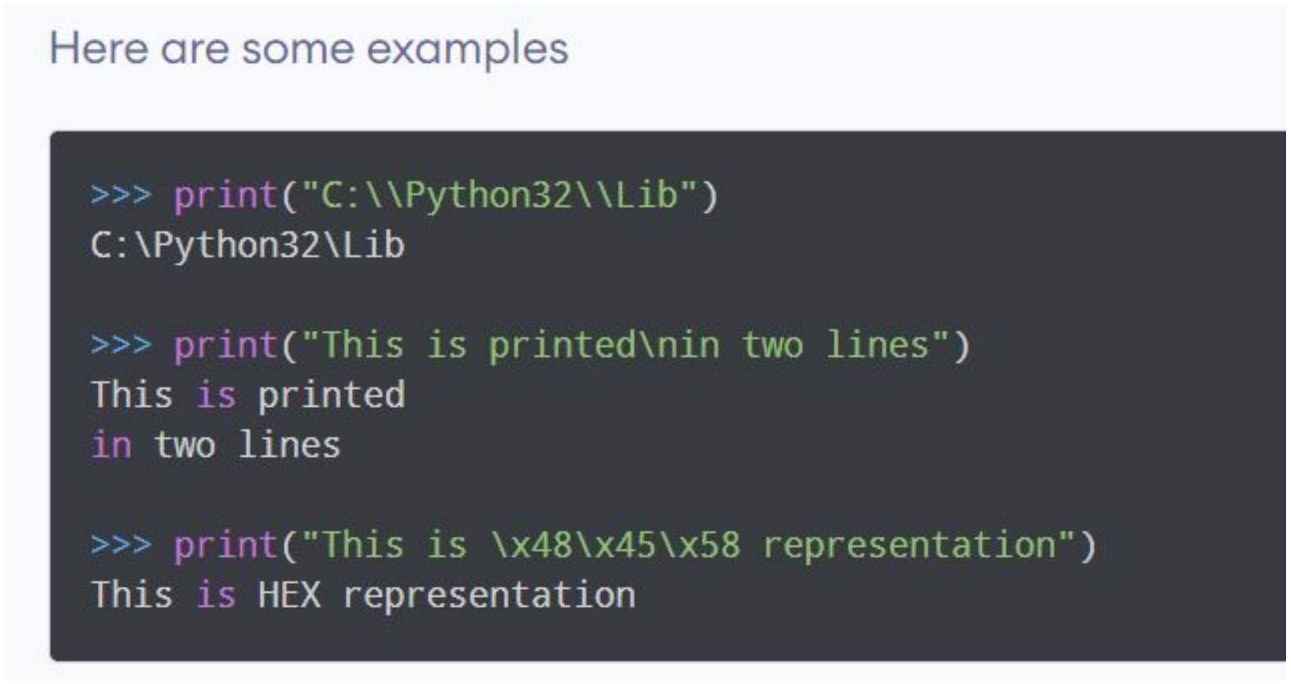
When we run the above program, we get the following output:

```
list(enumerate(str) =  [(0, 'c'), (1, 'o'), (2, 'l'), (3, 'd')]
len(str) =  4
```

Here is a list of all the escape sequences supported by Python.



| Escape Sequence | Description |
|---|---|
| \newline | Backslash and newline ignored |
| \\ | Backslash |
| \' | Single quote |
| \" | Double quote |
| \a | ASCII Bell |
| \b | ASCII Backspace |
| \f | ASCII Formfeed |
| \n | ASCII Linefeed |
| \r | ASCII Carriage Return |
| \t | ASCII Horizontal Tab |
| \v | ASCII Vertical Tab |
| \ooo | Character with octal value ooo |
| \xHH | Character with hexadecimal value HH |

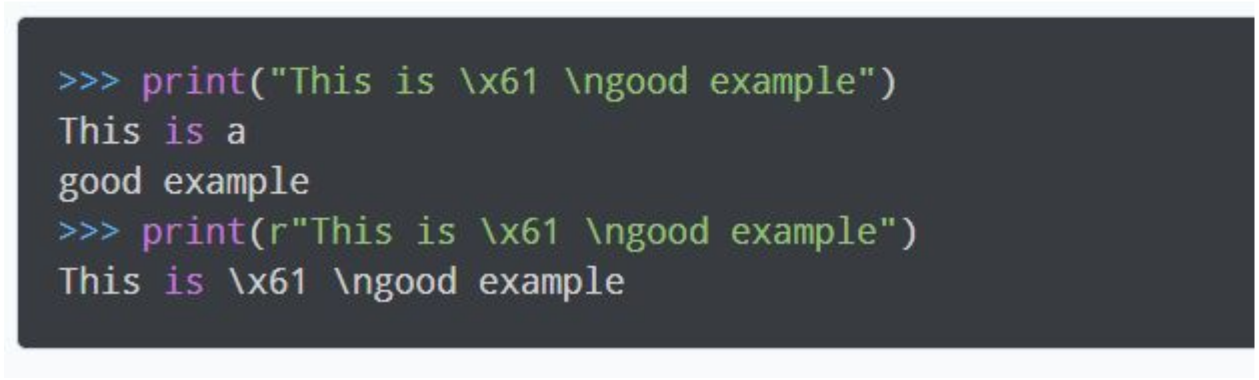


Here are some examples

```
>>> print("C:\\Python32\\Lib")
C:\Python32\Lib

>>> print("This is printed\nin two lines")
This is printed
in two lines

>>> print("This is \x48\x45\x58 representation")
This is HEX representation
```

## Raw String to ignore escape sequence

Sometimes we may wish to ignore the escape sequences inside a string. To do this we can place `r` or `R` in front of the string. This will imply that it is a raw string and any escape sequence inside it will be ignored.

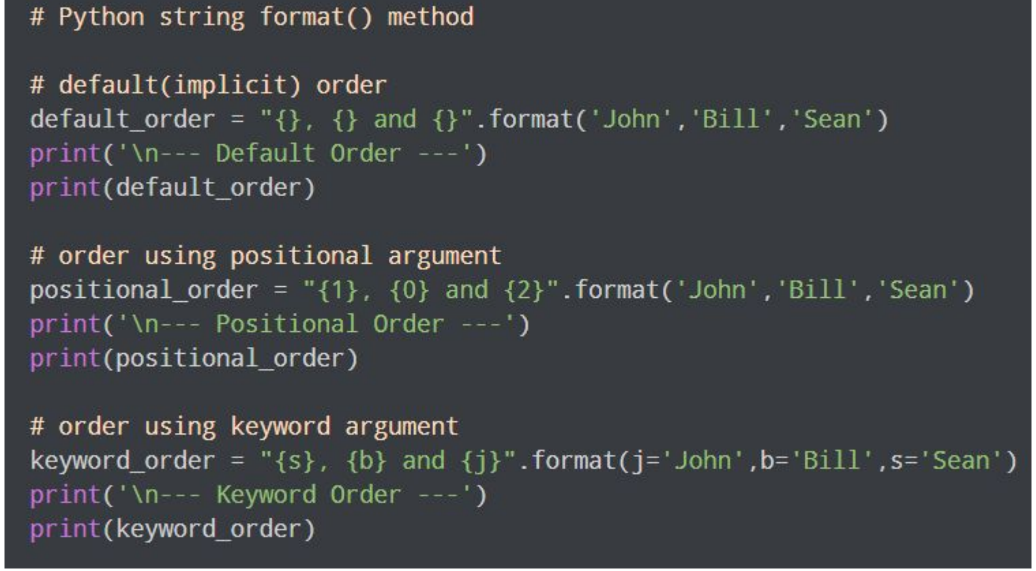


```
>>> print("This is \x61 \ngood example")
This is a
good example
>>> print(r"This is \x61 \ngood example")
This is \x61 \ngood example
```

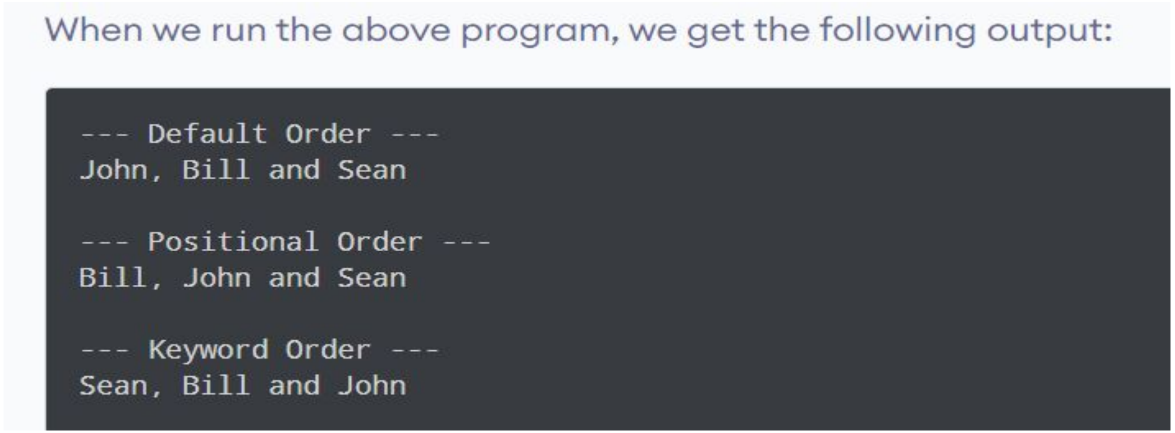
## The format() Method for Formatting Strings

The `format()` method that is available with the string object is very versatile and powerful in formatting strings. Format strings contain curly braces `{}` as placeholders or replacement fields which get replaced.

We can use positional arguments or keyword arguments to specify the order.



```python
# Python string format() method

# default(implicit) order
default_order = "{}, {} and {}".format('John','Bill','Sean')
print('\n--- Default Order ---')
print(default_order)

# order using positional argument
positional_order = "{1}, {0} and {2}".format('John','Bill','Sean')
print('\n--- Positional Order ---')
print(positional_order)

# order using keyword argument
keyword_order = "{s}, {b} and {j}".format(j='John',b='Bill',s='Sean')
print('\n--- Keyword Order ---')
print(keyword_order)
```



When we run the above program, we get the following output:

```
--- Default Order ---
John, Bill and Sean

--- Positional Order ---
Bill, John and Sean

--- Keyword Order ---
Sean, Bill and John
```

## Python String capitalize()

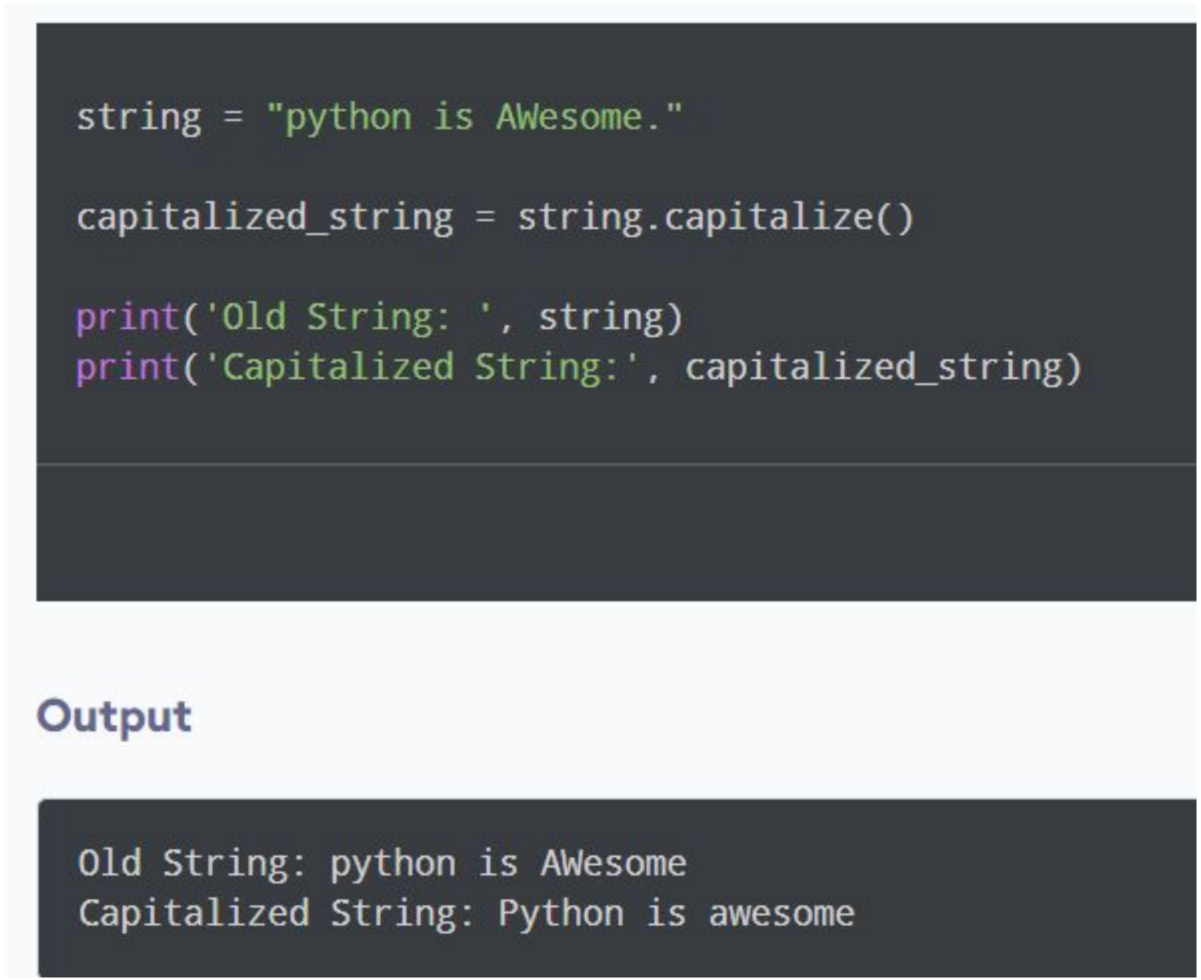
the capitalize() method converts first character of a string to uppercase letter and lowercases all other characters, if any.

The syntax of `capitalize()` is:

```
string.capitalize()
```

The `capitalize()` function doesn't take any parameter.

The `capitalize()` function returns a string with the first letter capitalized and all other characters lowercased. It doesn't modify the original string.



```python
string = "python is AWesome."

capitalized_string = string.capitalize()

print('Old String: ', string)
print('Capitalized String:', capitalized_string)
```

Output

```
Old String: python is AWesome
Capitalized String: Python is awesome
```

# Python String casefold()

The casefold() method is an aggressive lower() method which converts strings to case folded strings for caseless matching.

The `casefold()` method removes all case distinctions present in a string. It is used for caseless matching, i.e. ignores cases when comparing.

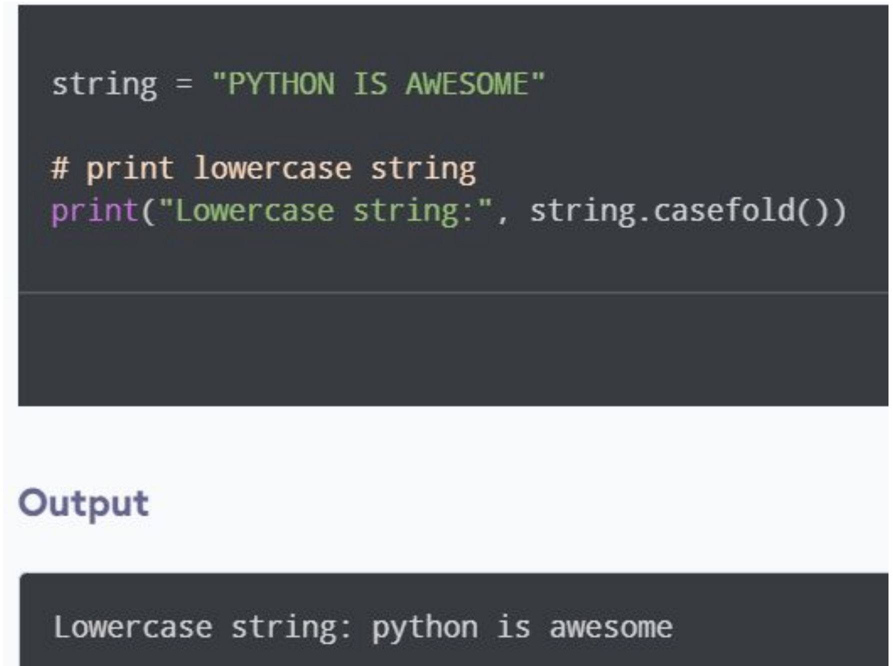
For example, the German lowercase letter `ß` is equivalent to `ss`. However, since `ß` is already lowercase, the `lower()` method does nothing to it. But, `casefold()` converts it to `ss`.

The syntax of `casefold()` is:

```
string.casefold()
```

The `casefold()` method doesn't take any parameters.

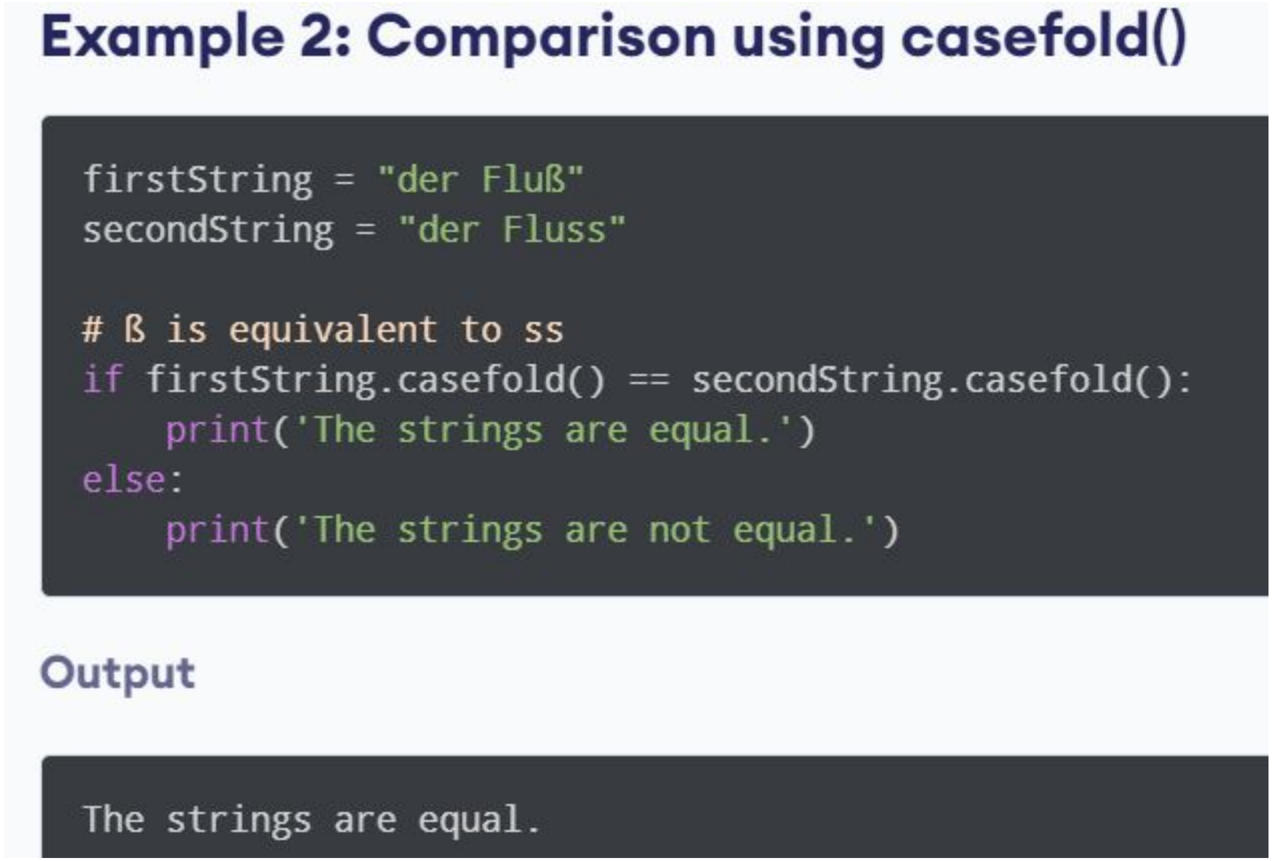
The `casefold()` method returns the case folded string.



```python
string = "PYTHON IS AWESOME"

# print lowercase string
print("Lowercase string:", string.casefold())
```

**Output**

```
Lowercase string: python is awesome
```



## Example 2: Comparison using casefold()

```python
firstString = "der Fluß"
secondString = "der Fluss"

# ß is equivalent to ss
if firstString.casefold() == secondString.casefold():
    print('The strings are equal.')
else:
    print('The strings are not equal.')
```

### Output

```
The strings are equal.
```

## Python String count()

The `count()` method returns the number of occurrences of a substring in the given string.

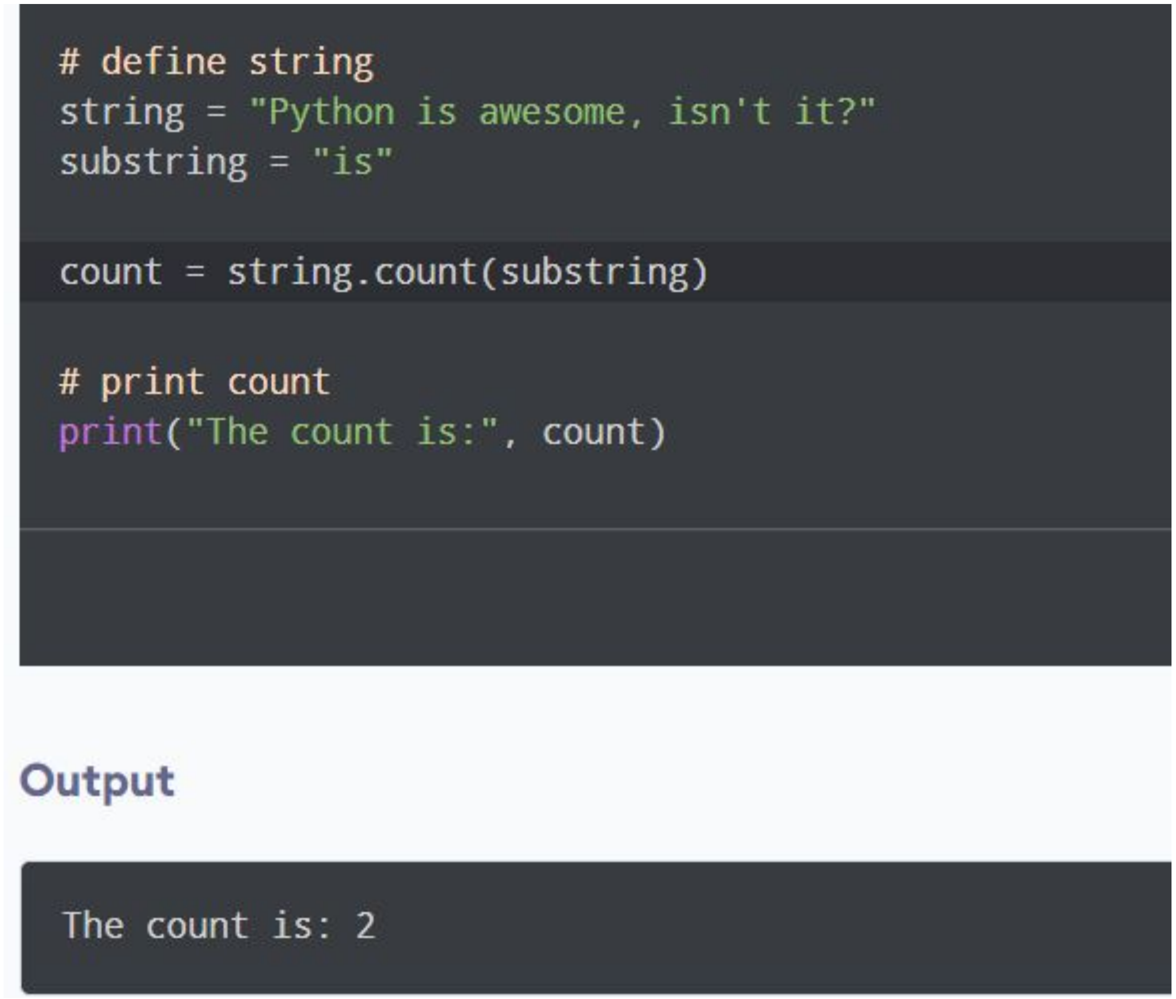
The syntax of `count()` method is:

```
string.count(substring, start=..., end=...)
```

**count() Parameters:**

`count()` method only requires a single parameter for execution. However, it also has two optional parameters:

- substring - string whose count is to be found.
- start (Optional) - starting index within the string where search starts.
- end (Optional) - ending index within the string where search ends.



```python
# define string
string = "Python is awesome, isn't it?"
substring = "is"

count = string.count(substring)

# print count
print("The count is:", count)
```

## Output

```
The count is: 2
```

## Example 2: Count number of occurrences of a given substring using start and end

```python
# define string
string = "Python is awesome, isn't it?"
substring = "i"

# count after first 'i' and before the last 'i'
count = string.count(substring, 8, 25)

# print count
print("The count is:", count)
```

Run Code »

### Output

```
The count is: 1
```

## Python String endswith()

The `endswith()` method returns `True` if a string ends with the specified suffix. If not, it returns `False`.

The syntax of `endswith()` is:

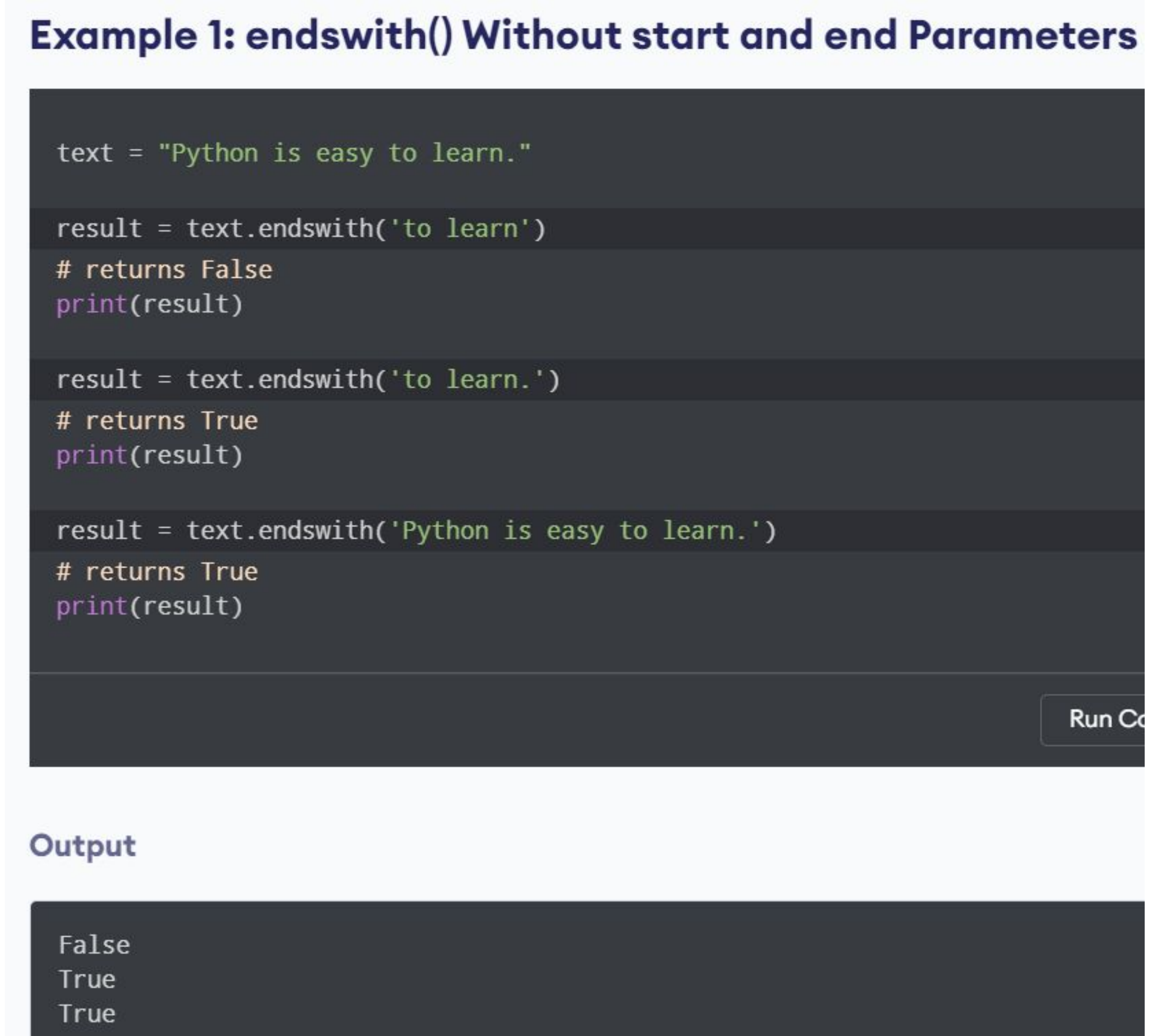
```
str.endswith(suffix[, start[, end]])
```

The `endswith()` takes three parameters:

- suffix - String or tuple of suffixes to be checked
- start (optional) - Beginning position where suffix is to be checked within the string.
- end (optional) - Ending position where suffix is to be checked within the string.

The `endswith()` method returns a boolean.

- It returns `True` if a string ends with the specified suffix.
- It returns `False` if a string doesn't end with the specified suffix.



## Example 1: endswith() Without start and end Parameters

```python
text = "Python is easy to learn."

result = text.endswith('to learn')
# returns False
print(result)

result = text.endswith('to learn.')
# returns True
print(result)

result = text.endswith('Python is easy to learn.')
# returns True
print(result)
```

Output

```
False
True
True
```

## Example 2: endswith() With start and end Parameters

```python
text = "Python programming is easy to learn."

# start parameter: 7
# "programming is easy to learn." string is searched
result = text.endswith('learn.', 7)
print(result)

# Both start and end is provided
# start: 7, end: 26
# "programming is easy" string is searched

result = text.endswith('is', 7, 26)
# Returns False
print(result)

result = text.endswith('easy', 7, 26)
# returns True
print(result)
```

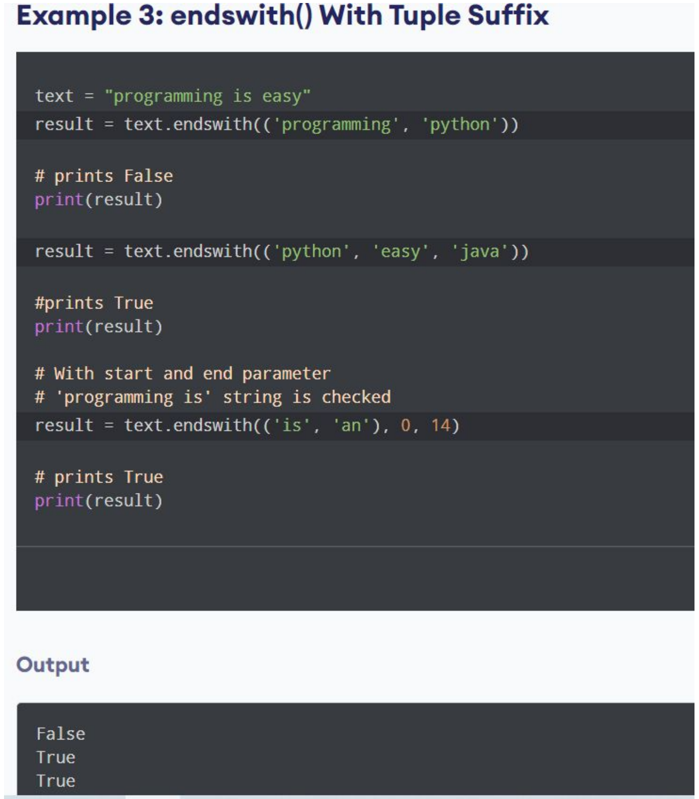
### Output

```
True
False
True
```

## Passing Tuple to endswith()

It's possible to pass a tuple suffix to the `endswith()` method in Python.

If the string ends with any item of the tuple, `endswith()` returns `True`. If not, it returns `False`



### Example 3: endswith() With Tuple Suffix

```python
text = "programming is easy"
result = text.endswith(('programming', 'python'))

# prints False
print(result)

result = text.endswith(('python', 'easy', 'java'))

#prints True
print(result)

# With start and end parameter
# 'programming is' string is checked
result = text.endswith(('is', 'an'), 0, 14)

# prints True
print(result)
```

**Output**

```
False
True
True
```

## Python String find()

The `find()` method returns the index of first occurrence of the substring (if found). If not found, it returns -1.

The syntax of the `find()` method is:

```
str.find(sub[, start[, end]] )
```

The `find()` method takes maximum of three parameters:

- sub - It is the substring to be searched in the `str` string.
- start and end (optional) - The range `str[start:end]` within which substring is searched.

The `find()` method returns an integer value:

- If the substring exists inside the string, it returns the index of the first occurence of the substring.
- If a substring doesn't exist inside the string, it returns -1.

## Example 1: find() With No start and end Argument

```python
quote = 'Let it be, let it be, let it be'

# first occurance of 'let it'(case sensitive)
result = quote.find('let it')
print("Substring 'let it':", result)

# find returns -1 if substring not found
result = quote.find('small')
print("Substring 'small ':", result)

# How to use find()
if (quote.find('be,') != -1):
    print("Contains substring 'be,'")
else:
    print("Doesn't contain substring")
```

### Output

```
Substring 'let it': 11
Substring 'small ': -1
Contains substring 'be,'
```

## Example 2: find() With start and end Arguments

```python
quote = 'Do small things with great love'

# Substring is searched in 'hings with great love'
print(quote.find('small things', 10))

# Substring is searched in ' small things with great love'
print(quote.find('small things', 2))

# Substring is searched in 'hings with great lov'
print(quote.find('o small ', 10, -1))

# Substring is searched in 'll things with'
print(quote.find('things ', 6, 20))
```

### Output

```
-1
3
-1
9
```

## Python String join()

The `join()` string method returns a string by joining all the elements of an iterable (list, string, tuple), separated by a string separator.

The syntax of the `join()` method is:

```
string.join(iterable)
```

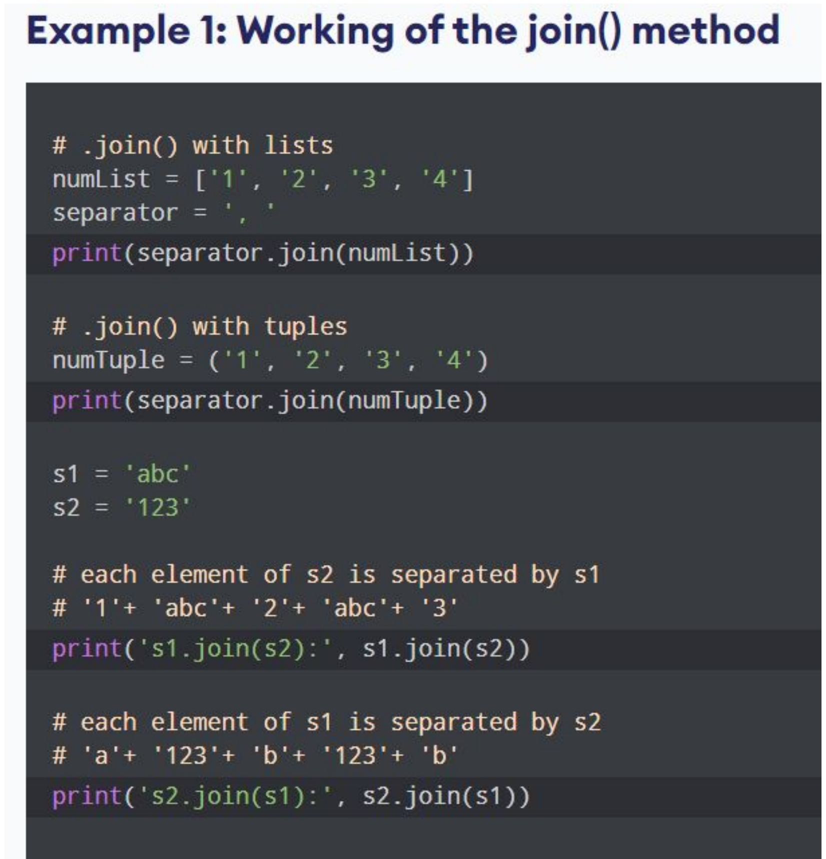
The `join()` method takes an iterable (objects capable of returning its members one at a time) as its parameter.

Some of the example of iterables are:

- Native data types - List, Tuple, String, Dictionary and Set.
- File objects and objects you define with an `__iter__()` or `__getitem()__` method.

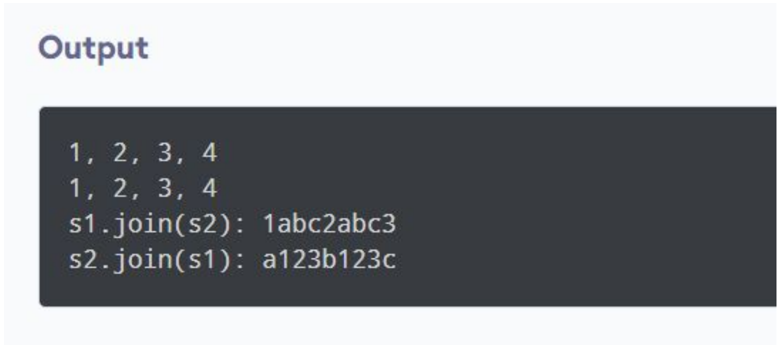
The `join()` method returns a string created by joining the elements of an iterable by string separator.

If the iterable contains any non-string values, it raises a `TypeError` exception.



## Example 1: Working of the join() method

```
# .join() with lists
numList = ['1', '2', '3', '4']
separator = ', '
print(separator.join(numList))

# .join() with tuples
numTuple = ('1', '2', '3', '4')
print(separator.join(numTuple))

s1 = 'abc'
s2 = '123'

# each element of s2 is separated by s1
# '1'+ 'abc'+ '2'+ 'abc'+ '3'
print('s1.join(s2):', s1.join(s2))

# each element of s1 is separated by s2
# 'a'+ '123'+ 'b'+ '123'+ 'b'
print('s2.join(s1):', s2.join(s1))
```



**Output**

```
1, 2, 3, 4
1, 2, 3, 4
s1.join(s2): 1abc2abc3
s2.join(s1): a123b123c
```

## Example 2: The join() method with sets

```python
# .join() with sets
test = {'2', '1', '3'}
s = ', '
print(s.join(test))

test = {'Python', 'Java', 'Ruby'}
s = '->->'
print(s.join(test))
```

### Output

```
2, 3, 1
Python->->Ruby->->Java
```

## Example 3: The join() method with dictionaries

```
# .join() with dictionaries
test = {'mat': 1, 'that': 2}
s = '->'

# joins the keys only
print(s.join(test))

test = {1: 'mat', 2: 'that'}
s = ', '

# this gives error since key isn't string
print(s.join(test))
```

### Output

```
mat->that
Traceback (most recent call last):
  File "...", line 12, in <module>
TypeError: sequence item 0: expected str instance, int found
```

# Python String replace()

The `replace()` method replaces each matching occurrence of the old character/text in the string with the new character/text.

It's syntax is:

```
str.replace(old, new [, count])
```

The `replace()` method can take maximum of 3 parameters:

- old - old substring you want to replace
- new - new substring which will replace the old substring
- count (optional) - the number of times you want to replace the `old` substring with the `new` substring

Note: If `count` is not specified, the `replace()` method replaces all occurrences of the `old` substring with the `new` substring.

The `replace()` method returns a copy of the string where the `old` substring is replaced with the `new` substring.

The original string is unchanged.

If the `old` substring is not found, it returns the copy of the original string.

## Example 1: Using replace()

```python
song = 'cold, cold heart'

# replacing 'cold' with 'hurt'
print(song.replace('cold', 'hurt'))

song = 'Let it be, let it be, let it be, let it be'

# replacing only two occurences of 'let'
print(song.replace('let', "don't let", 2))
```

### Output

```
hurt, hurt heart
Let it be, don't let it be, don't let it be, let it be
```

# Python String split()

The `split()` method breaks up a string at the specified separator and returns a list of strings.

The syntax of `split()` is:

```
str.split(separator, maxsplit)
```

## split() Parameters

The `split()` method takes a maximum of 2 parameters:

- separator (optional)- Delimiter at which splits occur. If not provided, the string is splitted at whitespaces.
- maxsplit (optional) - Maximum number of splits. If not provided, there is no limit on the number of splits.

The `split()` method returns a list of strings.

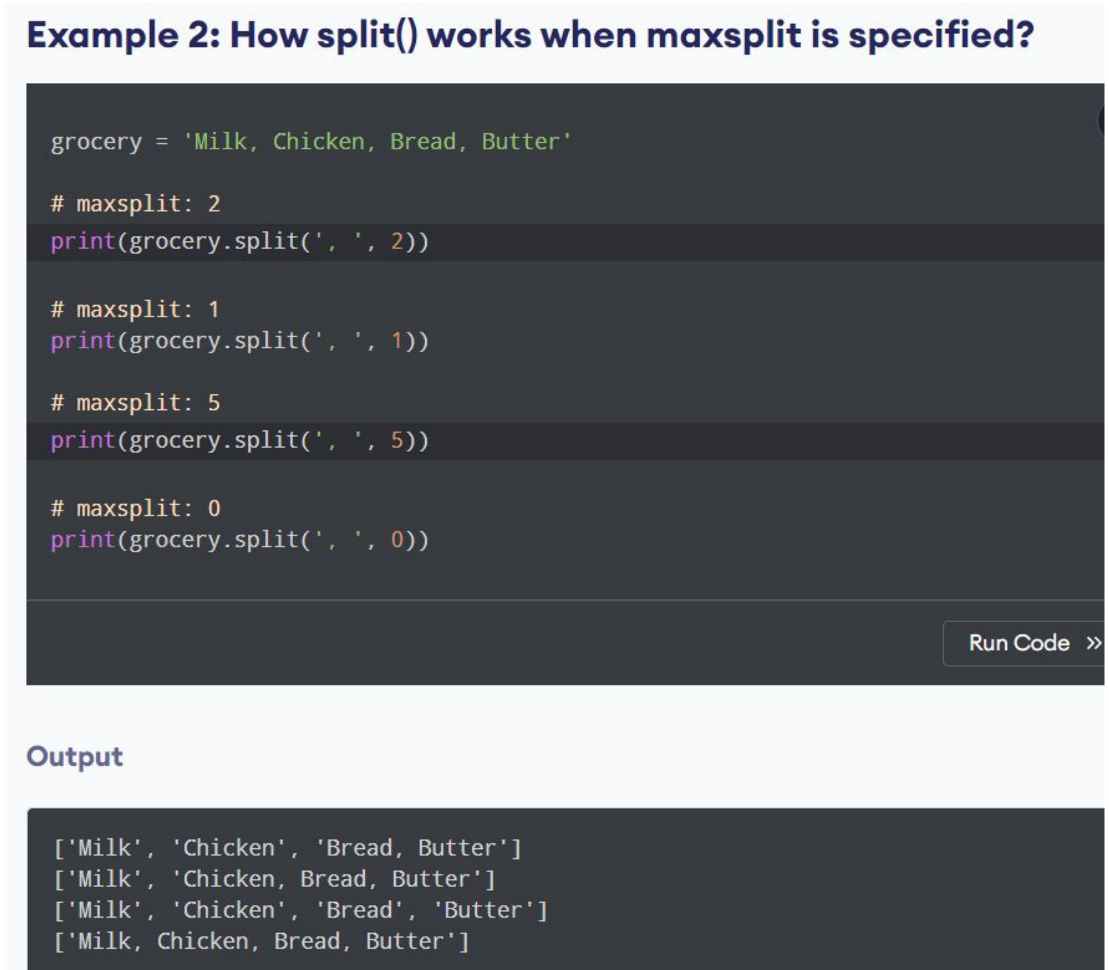
## Example 1: How split() works in Python?

```python
text= 'Love thy neighbor'

# splits at space
print(text.split())

grocery = 'Milk, Chicken, Bread'

# splits at ','
print(grocery.split(', '))

# Splits at ':'
print(grocery.split(':'))
```

### Output

```
['Love', 'thy', 'neighbor']
['Milk', 'Chicken', 'Bread']
['Milk, Chicken, Bread']
```

If `maxsplit` is specified, the list will have a maximum of `maxsplit+1` items.



## Example 2: How split() works when maxsplit is specified?

```
grocery = 'Milk, Chicken, Bread, Butter'

# maxsplit: 2
print(grocery.split(', ', 2))

# maxsplit: 1
print(grocery.split(', ', 1))

# maxsplit: 5
print(grocery.split(', ', 5))

# maxsplit: 0
print(grocery.split(', ', 0))
```

**Output**

```
['Milk', 'Chicken', 'Bread, Butter']
['Milk', 'Chicken, Bread, Butter']
['Milk', 'Chicken', 'Bread', 'Butter']
['Milk, Chicken, Bread, Butter']
```

## Python String upper()

The `upper()` method converts all lowercase characters in a string into uppercase characters and returns it.
The syntax of `upper()` method is:

```
string.upper()
```

`upper()` method doesn't take any parameters.

`upper()` method returns the uppercase string from the given string. It converts all lowercase characters to uppercase.

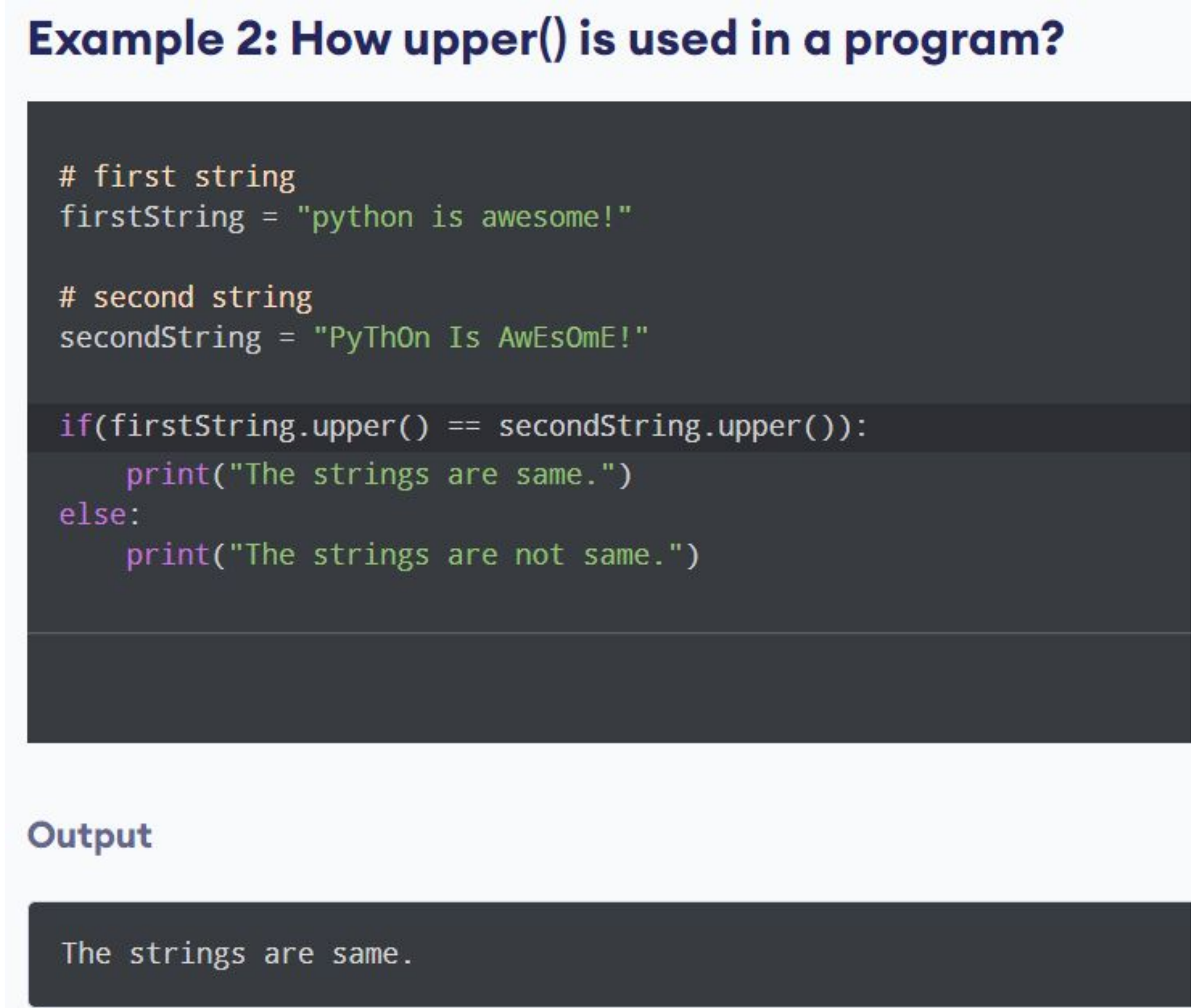
If no lowercase characters exist, it returns the original string.

## Example 1: Convert a string to uppercase

```python
# example string
string = "this should be uppercase!"
print(string.upper())

# string with numbers
# all alphabets should be lowercase
string = "Th!s ShOuLd B3 uPp3rCas3!"
print(string.upper())
```

### Output

```
THIS SHOULD BE UPPERCASE!
TH!S SHOULD B3 UPP3RCAS3!
```



## Example 2: How upper() is used in a program?

```python
# first string
firstString = "python is awesome!"

# second string
secondString = "PyThOn Is AwEsOmE!"

if(firstString.upper() == secondString.upper()):
    print("The strings are same.")
else:
    print("The strings are not same.")
```

### Output

```
The strings are same.
```