

Python String

A string is a sequence of characters. You must have used strings in other languages as well. Python strings play the same role as character arrays in languages like C, but they are somewhat higher-level tools than arrays.

Unlike languages such as C, in Python, strings come with a powerful set of processing tools.

Create a String

A python string is zero or more characters written inside single quotes `' '` or double quotes `" "`

```
S = 'Hello, World!' # single quotes
S = "Hello, World!" # double quotes
```

Multiline Strings

You can create a multiline string using triple-quotes: `""" """` or `''' '''`.

```
S = """String literals can
span multiple lines."""
print(S)

# String literals can
# span multiple lines.
```

The str() Constructor

You can convert almost any object in Python to a string using a type constructor called `str()`

```
# an integer to a string
S = str(42)
```

```

print(S)
# Prints '42'

# a complex number to a string
S = str(3+4j)
print(S)
# Prints '(3+4j)'

# a list to a string
S = str([1,1])
print(S)
# Prints '[1, 1]'

```

Access Characters by Index

You can access individual characters in a string using an index in square brackets. The string indexing starts from 0.

You can also access a string by negative indexing. A negative string index counts from the end of the string.

The indices for the elements in a string are illustrated as below:

0	1	2	3	4	5	6	7	8
A	B	C	D	E	F	G	H	I
-9	-8	-7	-6	-5	-4	-3	-2	-1

```

# Indexing
S = ' A B C D E F G H I '
print(S[0]) # Prints A
print(S[4]) # Prints E

```

```
# Negative Indexing
S = ' A B C D E F G H I '
print(S[-1]) # Prints I
print(S[-6]) # Prints D
```

Slicing a String

A segment of a string is called a slice and you can extract one by using a slice operator. A slice of a string is also a string.

The slice operator [n:m] returns the part of the string from the “n-th” item to the “m-th” item, including the first but excluding the last.

```
S = ' A B C D E F G H I '
print(S[2:5]) # Prints C D E
print(S[5:-1]) # Prints F G H
print(S[1:6:2]) # Prints B D F
```

The string slicing capability provided by python is extensive and covered in full detail [here](#).

Modify a String

It is tempting to use the [] operator on the left side of an assignment, in order to convert a character into a string. for example:

```
S = 'Hello, World!'
S[0] = 'J'
# Triggers TypeError: 'str' object does not support item assignment
```

The reason for the error is that the strings are unchangeable (immutable) and because of which you cannot change the existing string. The best you can do is create a new string that is a variation of the original:

```
S = 'Hello, world!'
new_S = 'J' + S[1:]
print(new_S)
```

```
# Prints Jello, world!
```

String Concatenation

You can concatenate strings using the concatenation operator `+` or the augmented assignment operator `+=`

```
# concatenation operator
```

```
S = 'Hello,' + ' World!'
```

```
print(S)
```

```
# Hello, World!
```

```
# augmented assignment operator
```

```
S = 'Hello,'
```

```
S += ' World!'
```

```
print(S)
```

```
# Prints Hello, World!
```

In Python, two or more strings next to each other are automatically concatenated, known as Implicit concatenation.

```
S = 'Hello,' " World!"
```

```
print(S)
```

```
# Prints Hello, World!
```

Implicit concatenation only works with two literals though, not with variables or expressions.

You can also put several strings within parentheses to join them together. This feature is useful when you want to break long strings.

```
S = ('Put strings within parentheses '
```

```
'to join them together.')
```

```
print(S)
```

```
# Put strings within parentheses to join them together.
```

You can replicate substrings in a string using the replication operator `*`

```
# the hard way
S = '-----'

# the easy way
S = '-' * 20
```

Find String Length

To find the number of characters in a string, use `len()` built-in function.

```
S = 'Supercalifragilisticexpialidocious'
print(len(S))
# Prints 34
```

Replace Text Within a String

Sometimes you want to replace a text inside a string, then you can use the `replace()` method.

```
S = 'Hello, World!'
x = S.replace('World', 'Universe')
print(x)
# Prints Hello, Universe!
```

Split and Join a String

Use `split()` method to chop up a string into a list of substrings, around a specified delimiter.

```
# Split the string on comma
S = 'red,green,blue,yellow'
x = S.split(',')
print(x)
# Prints ['red', 'green', 'blue', 'yellow']
```

```
print(x[0])  
# Prints red
```

And use `join()` method to join the list back into a string, with a specified delimiter in between.

```
# Join the list of substrings  
L = ['red', 'green', 'blue', 'yellow']  
S = ','.join(L)  
print(S)  
# Prints red,green,blue,yellow
```

String Case Conversion

Python provides five methods to perform case conversion on the target string viz. `lower()`, `upper()`, `capitalize()`, `swapcase()` and `title()`

```
S = 'Hello, World!'  
print(S.lower())  
# Prints hello, world!
```

```
S = 'Hello, World!'  
print(S.upper())  
# Prints HELLO, WORLD!
```

```
S = 'Hello, World!'  
print(S.capitalize())  
# Prints Hello, world!
```

```
S = 'Hello, World!'  
print(S.swapcase())  
# Prints hELLO, wORLD!
```

```
S = 'hello, world!'
print(S.title())
# Prints Hello, World!
```

Check if Substring Contains in a String

To check if a specific text is present in a string, use `in` operator. The `in` is a boolean operator, which takes two strings and returns `True` if the first appears as a substring in the second:

```
S = 'Hello, World!'
print('Hello' in S)
# Prints True
```

To search for a specific text within a string, use `find()` method. It returns the lowest index in the string where substring is found.

```
# Search for 'Foolish' within a string
S = 'Stay Hungry, Stay Foolish'
x = S.find('Foolish')
print(x)
# Prints 18
```

Iterate Through a String

To iterate over the characters of a string, use a simple `for loop`.

```
# Print each character in a string
S = 'Hello, World!'
for letter in S:
    print(letter, end=' ')
# Hello, World!
```

Python Escape Sequence

You can use quotes inside a string, as long as they don't match the quotes surrounding the string.

```
S = "We're open"          # Escape single quote
S = "I said 'Wow!'"      # Escape single quotes
S = 'I said "Wow!"'      # Escape double quotes
```

This is fine for most of the time but what if you want to declare a string with both single and double quotes like:

Bob told me, "Sam said, 'This won't work.'"

Python will raise a `SyntaxError`, because both quotation marks are special characters. The solution to avoid this problem is to use the backslash escape character `\`.

Prefixing a special character with `\` turns it into an ordinary character. This is called escaping.

```
S = "Bob told me, \"Sam said, 'This won't work.'\""
print(S)
# Prints Bob told me, "Sam said, 'This won't work.'"
```

Backslash escape character is used in representing certain special characters like: `\n` is a newline, `\t` is a tab. These are known as escape sequences.

```
S = str('First line.\n\tSecond line.')
print(S)
# First line.
#   Second line.
```

A list of all valid escape sequences in Python:

Escape Sequence

Meaning

`\newline`

Backslash and newline ignored

`\\`

Backslash (\)

`\'`

Single quote (')

`\"`

Double quote (")

`\a`

ASCII Bell (BEL)

`\b`

ASCII Backspace (BS)

`\f`

ASCII Formfeed (FF)

`\n` ASCII Linefeed (LF)

`\r` ASCII Carriage Return (CR)

`\t` ASCII Horizontal Tab (TAB)

`\v` ASCII Vertical Tab (VT)

`\ooo` Character with octal value ooo

`\xhh` Character with hex value hh

Raw String

Suppose you have stored a file path inside a string. When you execute it, you will get a result like this:

```
S = 'C:\new\text.txt'
```

```
print(S)
```

```
# C:
```

```
# ew      ext.txt
```

Here, `\n` is interpreted as newline and `\t` as tab.

If you don't want characters prefaced by `\` to be interpreted as special characters, you can declare the string as a raw string, by adding an `r` before the first quote.

```
S = r'C:\new\text.txt'
print(S)
# C:\new\text.txt
```

String Formatting/String Interpolation

In Python, there are three major ways to embed variables inside a string.

- printf-style % String Formatting
- str.format() Built-in Method
- f-String Formatter

```
# printf-style % string formatting
S = '%s is %d years old.' % ('Bob', 25)
print(S)
# Prints Bob is 25 years old.

# format() Built-in Method
S = '{1} is {0} years old.'.format(25, 'Bob')
print(S)
# Prints Bob is 25 years old.
```

The `format()` Built-in Method is covered in full detail [here](#).

```
# f-String Formatter
name = 'Bob'
age = 25
S = f'{name} is {age} years old.'
print(S)
# Prints Bob is 25 years old.
```

Python String Methods

Python has a set of built-in methods that you can call on string objects.

Method	Description
<code>capitalize()</code>	Capitalizes first character of the string
<code>casefold()</code>	Returns a casefolded string
<code>center()</code>	Returns center-aligned string
<code>count()</code>	Counts occurrences of a substring in a string
<code>encode()</code>	Return an encoded version of the string as a bytes object
<code>endswith()</code>	Determines whether the string ends with a given suffix
<code>expandtabs()</code>	Replaces tabs with spaces
<code>find()</code>	Searches the string for a given substring
<code>format()</code>	Perform a string formatting operation
<code>index()</code>	Searches the string for a given substring