String

In Python, string is an immutable sequence data type. It is the sequence of Unicode characters wrapped inside single, double, or triple quotes. There is a built-in class ‘str’ for handling Python strings. In python there is no data type to handle character.

Assigning values to a String

Strings can be created by enclosing a sequence of 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**.

```
>>> str1='This is a string'      # string in single quotes
>>> str2="This is 2nd string"    # string in double quotes
>>> str3= '''This is 3rd string''' # string in triple quotes
>>> str4='''This is
```

```
            the example of
            Multi-line string.
               '''
```

Output:

```
     >>> type(str1)
     Output: <class 'str'>
     >>> print(str1)
     Output: This is a string
     >>> print(str2)
     Output:This is 2nd string
     >>> print(str3)
     Output: This is 3rd string
     >>> print(str4)
     Output: This is
                the example of
                Multi-line string.


     >>>
```

If you need to use single quotes as a part of a string, delimit it with double quotes and vice versa.

```
     >>> a="Dogs are 'love'"
     >>> print(a)
```

Output: Dogs are ‘love’

```
     >>> b='"Dogs are" love'
     >>> print(b)
```

Output: “Dogs are” love

You can use as many quotes as you want, then.

```
>>> c='"Dogs ""are" "Love"'
>> print(c)
```

Output: "Dogs ""are" "Love"

However, you **cannot use a single quote to begin a string and a double quote to end it**, and **vice-versa**.

```
>>> a='Dogs are love"
```

SyntaxError: EOL while scanning string literal

Since we delimit strings using quotes, there are some things you need to take care of when using them inside a string.

```
>>> a="Dogs are "love""
```

SyntaxError: invalid syntax

Slicing a string

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring. **Syntax is str[Start_index:End_index:Stride]**. If **Start_index** is not provided, then the **default value** taken **is 0**. If **End_index** is not provided, then the **default value** taken **is last_index**. If **Stride** is not present, **default value is** taken as **1**. For example

```
>>> var1 = 'Hello World!'
>>> var2 = "Python Programming"
>>> print "var1[0]: ", var1[0]
```

```
>>> print "var2[1:5]: ", var2[1:5]
```

Output:

```
var1[0]:  H
var2[1:5]:  ytho
```

To print characters from the beginning to character 7

```
>>> print(var2[:8])
```

Output: Python P

```
>>> print(var2[:8:2])
```

Output: Pto r

To print characters from character 8 to the end of the string.

```
>>> print(var2[8:])
```

Output: rogramming

To print the entire string you can use

```
>>> print(var2[:])
```

Output: Python Programming

```
>>> print(var2[::3])
```

Output: Ph oai

You can also use a negative index. As usual last index is presented by -1. Therefore to print characters from the beginning to two characters less than the end of the string.

```
>>> print(var2[:-2])
```

Output: Python Programmi

To print characters from four characters from the end to the end of the string.

```
>>> var2[-4:]
```

Output: ming

To print characters from three characters from the string's end to two characters from the string's end .

```
>>> var2[-3:-2]
```

Output: i

To print characters from the end start at -1 go upto last index(- length of string) with a stride of -1. THat may be taken as default.

```
>>> var2[::-1]
```

Output: gnimmargorP nohtyP

The following codes return empty strings.

```
>>> var2[-2:-2]
```

Output: >>>

```
>>> var2[2:2]
```

Output: >>>

Since strings are arrays, we can loop through the characters in a string, with a for loop.

Loop through the letters in the word "banana":

```
for x in "banana":
    print(x)
```

Output: b

a

n

a

n

a

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
Output: 'Python'
>>> my_string = 'Programming'
>>> my_string
Output: Programming
```

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 Formatters

Sometimes, you may want to print variables along with a string. You can either use commas, or use string formatters for the same.

With comma (,)

```
>>> city='Kolaghat'
>>> college='CEM'
>>> print('You read in ',college,',',city)
```

Output: You read in CEM , Kolaghat

f-strings

The letter 'f' precedes the string, and the variables are mentioned in curly braces in their places.

```
>>> college='CEM, Kolaghat'
>>> estb=1998
>>> print(f"Welcome to {college} established in {estb}") #No space with in f and “”
Output: Welcome to CEM, Kolaghat established in 1998
```

format() method

You can use the format() method to do the same. It succeeds the string, and has the variables as arguments separated by commas. In the string, use curly braces to posit the variables. Inside the curly braces, you can either put 0,1,.. or the variables. When doing the latter, you must assign values to them in the format method.

```
>>> college='CEM,'
>>> city='Kolaghat'
>>> print('I Love {0},{1}'.format(college,city))
```

Output: I Love CEM,,Kolaghat

The variables don't have to be defined before the print statement.

```
>>> print("I love {a}".format(a=CEMK))
```
Output: I love CEMK

% operator

The % operator goes where the variables go in a string. %s is for string. What follows the string is the operator and variables in parentheses/in a tuple

The % operator goes where the variables go in a string. %s is for string. What follows the string is the operator and variables in parentheses/in a tuple.

```
>>> colg="CEMK"
>>> year=1998
>>> print("%s is established in %d"%(colg,year))
```
Output: CEMK is established in 1998

Other Options:

Here is the list of complete set of symbols which can be used along with % -

Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer

%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

Other supported symbols and functionality are listed in the following table −

Symbol	Functionality
*	argument specifies width or precision
-	left justification
+	display the sign
<sp>	leave a blank space before a positive number
#	add the octal leading zero ('0') or hexadecimal leading '0x' or '0X', depending on whether 'x' or 'X' were used.
0	pad from left with zeros (instead of spaces)
%	'%%' leaves you with a single literal '%'
(var)	mapping variable (dictionary arguments)
m.n.	m is the minimum total width and n is the number of digits to display after the decimal point (if appl.)

Escape Sequences in Python

In a Python string, you may want to put a tab, a linefeed, or other such things. Escape sequences allow us to do this. An escape sequence is a backslash followed by a character, depending on what you want to do. Python supports the following sequences.

- \n – line feed
- \t – tab
  >>> print("hell\to")
  Output: hell o

- \\ – backslash

Since a backslash may be a part of an escape sequence, so, a backslash must be escaped by a backslash too.

- \’ – A single quote can be escaped by a backslash. This lets you use single quotes freely in a string.
- \” – Like the single quote, the double quote can be escaped too.

Any Doubt yet in Python String and Python String Operations and Functions? Please Comment.

An escape character gets interpreted; in a single quoted as well as double quoted strings.

Backslash notation	Hexadecimal character	Description
\a	0x07	Bell or alert
\b	0x08	Backspace

\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed
\M-\C-x		Meta-Control-x
\n	0x0a	Newline
\nnn		Octal notation, where n is in the range 0to7
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x
\xnn		Hexadecimal notation, where n is in the range 0.9, a.f, or A.F

Python String Operations

Comparison

Python Strings can compare using the relational operators.

```
>>> 'hey'<'hi'
True
```

‘hey’ is lesser than ‘hi lexicographically (because i comes after e in the dictionary)

```
>>> a='check'
>>> a=='check'
True
>>> 'yes'!='no'
True
```

Arithmetic

Concatenation is the operation of joining stuff together. Python Strings can join using the concatenation operator +.

```
>>> a='Do you see this, '
>>> b='Dear Students!!'
>>> a+b
>>> print(c)
Output: Do you see this, Dear Students!!
```

Some arithmetic operations can be applied on strings. Multiplying ‘na’ by 2 returned ‘nana’ , and not 2xna, because ‘a’ is a string, not a number.

```
>>> 'ba'+'na'*2
Output: ‘banana’
>>> a=’10’
>>> print(2*a)
Output: 1010
```

You **cannot concatenate a string to a number.**

```
>>> '10'+10
Traceback (most recent call last):
File “<pyshell#49>”, line 1, in <module>
’10’+10
```

```
TypeError: must be str, not int
```

Membership

The membership operators of Python can be used to check if string is a substring to another.

```
>>> 'na' in 'banana'
True
>>> 'less' not in 'helpless'
False
```

Identity

Python’s identity operators ‘is’ and ‘is not’ can be used on strings.

```
>>> 'Hey' is 'Hi'
False
>>> 'Yo' is not 'yo'
True
```

Logical

Python’s and, or, and not operators can be applied too. An **empty string has a Boolean value of False**.

and : If the value on the left is True it returns the value on the right. Otherwise, the value on the left is False, it returns False.

```
>>> '' and '1'   # False and True ⇒ False
```

Output: ‘’

>>> '1' and ''

Output: ”

or: If the value on the left is True, it returns True. Otherwise, the value on the right is returned.

>>> '1' or ""

Output: '1'

not : As we said earlier, an empty string has a Boolean value of False.

>>> not('1')

False

>>> not('')

True

This was all about the tutorial on Python strings. Hope you like the Python strings tutorial.

Python String Functions

Python provides us with a number of functions that we can apply on strings or to create strings.

len()

The len() function returns the length of a string.

>>> a='CEMK'

>>> len(a)

Output: 4

>>> len(a[2:])

Output: 2

str()

This function converts any data type into a string.

```
>>> x=str(2+3j)
>>> x
Output: '(2+3j)'
>>> type(x)
Output: <class 'str'>
>>> str(['red','green','blue'])
Output: "['red', 'green', 'blue']"
```

sorted()

It sorts all the string in a lexicography order.

```
>>> var2=['Python', 'Programming','aaaa','Zzzzz']
>>> sorted(var2)
Output: ['Programming', 'Python', 'Zzzzz', 'aaaa']
>>> sorted(var2,reverse=True)
Output: ['aaaa', 'Zzzzz', 'Python', 'Programming']
```

Python String Methods

index()/find()

The index() or find() method returns the index of the first occurrence of the sub-string in the given string. If substring is not present find() returns -1 while index() returns Error.

```
>>> var2="Python Programming"
>>> var2.index('o')       or var2.find('o')
Output; 4
>>> var2.index('m')        or var2.find('m')
Output: 13
>>> var2.index('ing')      or var2.find('ing')
```

```
Output: 15
>>> var2.find('www')          Output: -1
>>> var2.index('www')    Output: ValueError: substring not found
```

lower() and upper()

These methods return the string in lowercase and uppercase, respectively.

```
>>> a='Book'
>>> a.lower()
Output: 'book'
>>> a.upper()
Output: 'BOOK'
```

strip()

It removes whitespaces from the beginning and end of the string.

```
>>> a="  Book     "
>>> a.strip()
Output: "Book"
```

isdigit()

Returns True if all characters in a string are digits.

```
>>> a='777'
>>> a.isdigit()
Output: True
>>> a='77a'
>>> a.isdigit()
```

```
Output: False
```

isalpha()

Returns True if all characters in a string are characters from an alphabet.

```
>>> a='abc'
>>> a.isalpha()
Output: True
>>> a='ab7'
>>> a.isalpha()
Output: False
```

isspace()

Returns True if all characters in a string are spaces.

```
>>> a='   '
>>> a.isspace()
True
>>> a=' \'  '
>>> a.isspace()
Output: False
>>> a='    \t'
>>> a.isspace()
Output: True
```

startswith()

It takes a string as an argument, and returns True is the string it is applied on begins with the string in the argument.

```
>>> a='cemk'
>>> a.startswith('ce')
Output: True
>>> a.startswith('CE')
Output: False
```

endswith()

It takes a string as an argument, and returns True if the string it is applied on ends with the string in the argument.

```
>>> a='therefore'
>>> a.endswith('fore')
Output: True
```

find()

It takes an argument and searches for it in the string on which it is applied. It then returns the starting index of the substring.

```
>>> 'homeowner'.find('meow')
Output: 2
>>> a='College of Engineering and Management, Kolaghat'
>>> a.find('ring')
Output: 18
```

If the string doesn’t exist in the main string, then the index it returns is -1.

>>> 'homeowner'.find('wow')

Output: -1

replace()

It takes two arguments. The first is the substring to be replaced. The second is the substring to replace with.

>>> 'banana'.replace('na','ha')

Output: ‘bahaha’

split()

It takes one argument. The string is then split around every occurrence of the argument in the string.

>>> 'No. Okay. Why?'.split('.')

Output: [‘No’, ‘ Okay’, ‘ Why?’]

>>> b='College.of.Engineering.and.Management,.Kolaghat'

>>> a=b.split('.')

>>> print(a)

Output: ['College', 'of', 'Engineering', 'and', 'Management,', 'Kolaghat']

>>> print(a[0])

Output: College

>>> print(a[3])

Output: and

join( )

It takes a list as an argument and joins the elements in the list using the string it is applied on.

```
>>> "*".join(['red','green','blue'])
```

Output: ‘red*green*blue’

```
>>> a=['College', 'of', 'Engineering', 'and', 'Management,', 'Kolaghat']
>>> jn=" ".join(a)
>>> print(jn)
```

Output: College of Engineering and Management, Kolaghat

Python includes the following built-in methods to manipulate strings −

Sr.No.	Methods with Description
1	**capitalize():**Capitalizes first letter of string
2	**center(width, fillchar):** Returns a space-padded string with the original string centered to a total of width columns.
3	**count(str, beg= 0,end=len(string)):** Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
4	**decode(encoding='UTF-8',errors='strict'):** Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.
5	**encode(encoding='UTF-8',errors='strict'):** Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.

6	**endswith(suffix, beg=0, end=len(string)):** Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.
7	**expandtabs(tabsize=8):** Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided.
8	**find(str, beg=0 end=len(string)):** Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
9	**index(str, beg=0, end=len(string)):** Same as find(), but raises an exception if str not found.
10	**isalnum() :** Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
11	**isalpha():** Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
12	**isdigit():** Returns true if string contains only digits and false otherwise.
13	**islower():** Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
14	**isnumeric():** Returns true if a unicode string contains only numeric characters and false otherwise.
15	**isspace**(): Returns true if string contains only whitespace characters and false otherwise.
16	**istitle():** Returns true if string is properly "titlecased" and false otherwise.

17	**isupper():** Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
18	**join(seq):** Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
19	**len(string):** Returns the length of the string
20	**ljust(width[, fillchar]):** Returns a space-padded string with the original string left-justified to a total of width columns.
21	**lower():** Converts all uppercase letters in string to lowercase.
22	**lstrip():** Removes all leading whitespace in string.
23	**maketrans():** Returns a translation table to be used in translate function.
24	**max(str):** Returns the max alphabetical character from the string str.
25	**min(str):** Returns the min alphabetical character from the string str.
26	**replace(old, new [, max]):** Replaces all occurrences of old in string with new or at most max occurrences if max given.
27	**rfind(str, beg=0,end=len(string)):** Same as find(), but search backwards in string.
28	**rindex(str, beg=0, end=len(string)):** Same as index(), but search backwards in string.

29	**rjust(width,[, fillchar]):** Returns a space-padded string with the original string right-justified to a total of width columns.
30	**rstrip() :** Removes all trailing whitespace of string.
31	**split(str="", num=string.count(str)):** Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.
32	**splitlines(num=string.count('\n')):** Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed.
33	**startswith(str, beg=0,end=len(string)):** Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; returns true if so and false otherwise.
34	**strip([chars]):** Performs both lstrip() and rstrip() on string.
35	**swapcase():** Inverts case for all letters in string.
36	**title():** Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase.
37	**translate(table, deletechars="") :** Translates string according to translation table str(256 chars), removing those in the del string.
38	**upper():** Converts lowercase letters in string to uppercase.
39	**zfill (width)** : Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).

40	**isdecimal()** : Returns true if a unicode string contains only decimal characters and false otherwise.