

PYTHON STRINGS

http://www.tutorialspoint.com/python/python_strings.htm

Copyright © tutorialspoint.com

Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes.

Creating strings is as simple as assigning a value to a variable. For example:

```
var1 = 'Hello World!'
var2 = "Python Programming"
```

Accessing Values in Strings:

Python does not support a character type; these are treated as strings of length one, thus also considered a substring.

To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring. Following is a simple example:

```
#!/usr/bin/python

var1 = 'Hello World!'
var2 = "Python Programming"

print "var1[0]: ", var1[0]
print "var2[1:5]: ", var2[1:5]
```

When the above code is executed, it produces the following result:

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

Updating Strings:

You can "update" an existing string by (re)assigning a variable to another string. The new value can be related to its previous value or to a completely different string altogether. Following is a simple example:

```
#!/usr/bin/python

var1 = 'Hello World!'

print "Updated String :- ", var1[:6] + 'Python'
```

When the above code is executed, it produces the following result:

```
Updated String :-  Hello Python
```

Escape Characters:

Following table is a list of escape or non-printable characters that can be represented with backslash notation.

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 | ox1b | Escape |
| \f | ox0c | Formfeed |
| \M-\C-x | | Meta-Control-x |
| \n | ox0a | Newline |
| \nnn | | Octal notation, where n is in the range 0-7 |
| \r | ox0d | Carriage return |
| \s | ox20 | Space |
| \t | ox09 | Tab |
| \v | ox0b | Vertical tab |
| \x | | Character x |
| \xnn | | Hexadecimal notation, where n is in the range 0-9, a-f, or A-F |

String Special Operators:

Assume string variable a holds 'Hello' and variable b holds 'Python', then:

| Operator | Description | Example |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------|
| + | Concatenation - Adds values on either side of the operator | a + b will give HelloPython |
| * | Repetition - Creates new strings, concatenating multiple copies of the same string | a*2 will give -HelloHello |
| [] | Slice - Gives the character from the given index | a[1] will give e |
| [:] | Range Slice - Gives the characters from the given range | a[1:4] will give ell |
| in | Membership - Returns true if a character exists in the given string | H in a will give 1 |
| not in | Membership - Returns true if a character does not exist in the given string | M not in a will give 1 |
| r/R | Raw String - Suppresses actual meaning of Escape characters. The syntax for raw strings is exactly the same as for normal strings with the exception of the raw string operator, the letter "r," which precedes the quotation marks. The "r" can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote mark. | print r'\n' prints \n and print R'\n' prints \n |
| % | Format - Performs String formatting | See at next section |

String Formatting Operator:

One of Python's coolest features is the string format operator %. This operator is unique to strings and makes up for the lack of having functions from C's printf() family. Following is a simple example:

```
#!/usr/bin/python

print "My name is %s and weight is %d kg!" % ('Zara', 21)
```

When the above code is executed, it produces the following result:

```
My name is Zara and weight is 21 kg!
```

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 ('o') or hexadecimal leading 'ox' or 'oX', 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.) |

Triple Quotes:

Python's triple quotes comes to the rescue by allowing strings to span multiple lines, including verbatim NEWLINES, TABs, and any other special characters.

The syntax for triple quotes consists of three consecutive **single or double** quotes.

```
#!/usr/bin/python

para_str = """this is a long string that is made up of
several lines and non-printable characters such as
TAB ( \t ) and they will show up that way when displayed.
NEWLINES within the string, whether explicitly given like
this within the brackets [ \n ], or just a NEWLINE within
the variable assignment will also show up.
"""
print para_str;
```

When the above code is executed, it produces the following result. Note how every single special character has been converted to its printed form, right down to the last NEWLINE at the end of the string between the "up." and closing triple quotes. Also note that NEWLINES occur either with an explicit carriage return at the end of a line or its escape code (\n):

```
this is a long string that is made up of
several lines and non-printable characters such as
TAB (    ) and they will show up that way when displayed.
NEWLINES within the string, whether explicitly given like
this within the brackets [
    ], or just a NEWLINE within
the variable assignment will also show up.
```

Raw strings don't treat the backslash as a special character at all. Every character you put into a raw string stays the way you wrote it:

```
#!/usr/bin/python

print 'C:\\nowhere'
```

When the above code is executed, it produces the following result:

```
C:\nowhere
```

Now let's make use of raw string. We would put expression in **r'expression'** as follows:

```
#!/usr/bin/python

print r'C:\\nowhere'
```

When the above code is executed, it produces the following result:

```
C:\\nowhere
```

Unicode String:

Normal strings in Python are stored internally as 8-bit ASCII, while Unicode strings are stored as 16-bit Unicode. This allows for a more varied set of characters, including special characters from most languages in

the world. I'll restrict my treatment of Unicode strings to the following:

```
#!/usr/bin/python
print u'Hello, world!'
```

When the above code is executed, it produces the following result:

```
Hello, world!
```

As you can see, Unicode strings use the prefix `u`, just as raw strings use the prefix `r`.

Built-in String Methods:

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

| SN | Methods with Description |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | <code>capitalize()</code> Capitalizes first letter of string |
| 2 | <code>center(width, fillchar)</code> Returns a space-padded string with the original string centered to a total of width columns |
| 3 | <code>count(str, beg = 0, end = len(string))</code> Counts how many times <code>str</code> occurs in <code>string</code> or in a substring of <code>string</code> if starting index <code>beg</code> and ending index <code>end</code> are given |
| 4 | <code>decode(encoding='UTF-8', errors='strict')</code> Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding. |
| 5 | <code>encode(encoding='UTF-8', errors='strict')</code> Returns encoded string version of <code>string</code> ; on error, default is to raise a <code>ValueError</code> unless <code>errors</code> is given with 'ignore' or 'replace'. |
| 6 | <code>endswith(suffix, beg=0, end=len(string))</code> Determines if <code>string</code> or a substring of <code>string</code> (if starting index <code>beg</code> and ending index <code>end</code> are given) ends with <code>suffix</code> ; returns true if so and false otherwise |
| 7 | <code>expandtabs(tabsize=8)</code> Expands tabs in <code>string</code> to multiple spaces; defaults to 8 spaces per tab if <code>tabsize</code> not provided |
| 8 | <code>find(str, beg=0 end=len(string))</code> Determine if <code>str</code> occurs in <code>string</code> or in a substring of <code>string</code> if starting index <code>beg</code> and ending index <code>end</code> are given returns index if found and -1 otherwise |
| 9 | <code>index(str, beg=0, end=len(string))</code> Same as <code>find()</code> , but raises an exception if <code>str</code> not found |
| 10 | <code>isalnum()</code> Returns true if <code>string</code> has at least 1 character and all characters are alphanumeric and false otherwise |
| 11 | <code>isalpha()</code> Returns true if <code>string</code> has at least 1 character and all characters are alphabetic and false otherwise |
| 12 | <code>isdigit()</code> Returns true if <code>string</code> contains only digits and false otherwise |
| 13 | <code>islower()</code> Returns true if <code>string</code> has at least 1 cased character and all cased characters are in lowercase and false otherwise |

| | |
|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 14 | <u>isnumeric()</u> Returns true if a unicode string contains only numeric characters and false otherwise |
| 15 | <u>isspace()</u> Returns true if string contains only whitespace characters and false otherwise |
| 16 | <u>istitle()</u> Returns true if string is properly "titlecased" and false otherwise |
| 17 | <u>isupper()</u> Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise |
| 18 | <u>join(seq)</u> Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string |
| 19 | <u>len(string)</u> Returns the length of the string |
| 20 | <u>ljust(width[, fillchar])</u> Returns a space-padded string with the original string left-justified to a total of width columns |
| 21 | <u>lower()</u> Converts all uppercase letters in string to lowercase |
| 22 | <u>lstrip()</u> Removes all leading whitespace in string |
| 23 | <u>maketrans()</u> Returns a translation table to be used in translate function. |
| 24 | <u>max(str)</u> Returns the max alphabetical character from the string str |
| 25 | <u>min(str)</u> Returns the min alphabetical character from the string str |
| 26 | <u>replace(old, new[, max])</u> Replaces all occurrences of old in string with new or at most max occurrences if max given |
| 27 | <u>rfind(str, beg=0, end=len(string))</u> Same as find(), but search backwards in string |
| 28 | <u>rindex(str, beg=0, end=len(string))</u> Same as index(), but search backwards in string |
| 29 | <u>rjust(width[, fillchar])</u> Returns a space-padded string with the original string right-justified to a total of width columns. |
| 30 | <u>rstrip()</u> Removes all trailing whitespace of string |
| 31 | <u>split(str="", num=string.count(str))</u> Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given |
| 32 | <u>splitlines(num=string.count('\n'))</u> Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed |
| 33 | <u>startswith(str, beg=0, end=len(string))</u> 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 | <u>strip([chars])</u> |

| | |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | Performs both <code>lstrip()</code> and <code>rstrip()</code> on string |
| 35 | <code>swapcase()</code> Inverts case for all letters in string |
| 36 | <code>title()</code> Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase |
| 37 | <code>translate(table, deletechars="")</code> Translates string according to translation table str(256 chars), removing those in the delstring |
| 38 | <code>upper()</code> Converts lowercase letters in string to uppercase |
| 39 | <code>zfill(width)</code> Returns original string leftpadded with zeros to a total of width characters; intended for numbers, <code>zfill()</code> retains any sign given (less one zero) |
| 40 | <code>isdecimal()</code> Returns true if a unicode string contains only decimal characters and false otherwise |