012 Python String

String

Python has a built-in string class named "str" with many useful features. String literals can be enclosed by either single or double, although single quotes are more commonly used.

Backslash escapes work the usual way within both single and double quoted literals -- e.g. \n \' \". A double quoted string literal can contain single quotes without any fuss (e.g. "I wouldn't be a painter") and likewise single quoted string can contain double quotes.

A string literal can span multiple lines, use triple quotes to start and end them. You can use single quotes too, but there must be a backslash \ at the end of each line to escape the newline.

String: Commands

```
# Strips all whitespace characters from both ends.
<str>  = <str>.strip()
# Strips all passed characters from both ends.
<str>  = <str>.strip('<chars>')

# Splits on one or more whitespace characters.
<list> = <str>.split()
# Splits on 'sep' str at most 'maxsplit' times.
<list> = <str>.split(sep=None, maxsplit=-1)
# Splits on line breaks. Keeps them if 'keepends'.
<list> = <str>.splitlines(keepends=False)
# Joins elements using string as separator.
<str>  = <str>.join(<coll_of_strings>)

# Checks if string contains a substring.
<bool> = <sub_str> in <str>
# Pass tuple of strings for multiple options.
<bool> = <str>.startswith(<sub_str>)
# Pass tuple of strings for multiple options.
<bool> = <str>.endswith(<sub_str>)
# Returns start index of first match or -1.
<int>  = <str>.find(<sub_str>)
```

```
# Same but raises ValueError if missing.
<int>  = <str>.index(<sub_str>)

# Replaces 'old' with 'new' at most 'count' times.
<str>  = <str>.replace(old, new [, count])
# True if str contains only numeric characters.
<bool> = <str>.isnumeric()
# Nicely breaks string into lines.
<list> = textwrap.wrap(<str>, width)
```

- **Also: 'lstrip()', 'rstrip()'.**
- **Also: 'lower()', 'upper()', 'capitalize()' and 'title()'.**

Char

```
# Converts int to unicode char.
<str> = chr(<int>)
# Converts unicode char to int.
<int> = ord(<str>)
>>> ord('0'), ord('9')
(48, 57)
>>> ord('A'), ord('Z')
(65, 90)
ord('a'), ord('z')
(97, 122)
```

Initialize string literals in Python:

```
>>> s = "Python string"

>>> print(s)

Python string

>>> #using single quote and double quote

>>> s = "A Poor Woman's Journey."

>>> print(s)

A Poor Woman's Journey.

>>> s = "I read the article, 'A Poor Woman's Journey.'"

>>> print(s)

I read the article, 'A Poor Woman's Journey.'
```

```
>>> s = 'dir "c:\&temp\*.sas" /o:n /b > "c:\&temp\abc.txt"'

>>> print(s)

dir "c:\&temp\*.sas" /o:n /b > "c:\&temp bc.txt"

>>> #print multiple lines

>>> s = """jQuery exercises

        JavaScript tutorial

        Python tutorial and exercises ..."""

>>> print(s)

jQuery exercises

        JavaScript tutorial

        Python tutorial and exercises ...

>>> s = 'jQuery exercises\n  JavaScript tutorial\n Python tutorial and
exercises ...'

>>> print(s)

jQuery exercises

         JavaScript tutorial

           Python tutorial and exercises ...

>>>
```

Access character(s) from a string:

Characters in a string can be accessed using the standard [ ] syntax, and like Java and C++, Python uses zero-based indexing, so if str is 'hello' str[2] is 'l'. If the index is out of bounds for the string, Python raises an error.

```
>>> a = "Python string"

>>> print (a)

Python string
```

```
>>> b = a[2]

>>> print(b)

t

>>> a[0]

'P'

>>>
```

Explanation :

- The above statement selects character at position number 2 from a and assigns it to b.
- The expression in brackets is called an index that indicates which character we are interested.
- The index is an offset from the beginning of the string, and the offset of the first letter is zero.

We can use any expression, including variables and operators, as an index

```
>>> a = "Python string"

>>> b = a[4+3]

>>> print(b)

s

>>>
```

The value of the index has to be an integer.

```
>>> a = "Python string"

>>> a[2.3]

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

TypeError: string indices must be integers

>>>
```

Negative indices:

Alternatively, we can use negative indices, which count backward from the end of the

Index -1 gives the last character.

```
>>> a = "Python string"

>>> a[-2]

'n'

>>> a[-8]

'n'

>>>
```

Python strings are immutable:

- Python strings are "immutable" which means they cannot be changed after they are created
- Therefore we can not use [ ] operator on the left side of an assignment.
- We can create a new string that is a variation of the original.

```
>>> a = "PYTHON"

>>> a[0] = "x"

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

TypeError: 'str' object does not support item assignment

>>> a = "PYTHON"

# Concatenates a new first letter on to a slice of greeting.

>>> b = "X" + a[1:]

>>> print(b)
```

```
XYTHON
# It has no effect on the original string.
>>> print(a)
PYTHON
>>>
```

Python String concatenation:

The '+' operator is used to concatenate two strings.

```
>>> a = "Python" + "String"
>>> print(a)
PythonString
>>>
```

You can also use += to concatenate two strings.

```
>>> a = "Java"
>>> b = "Script"
>>> a+=b
>>> print(a)
JavaScript
>>>
```

Using '*' operator:

```
>>> a = "Python" + "String"
>>> b = "<" + a*3 + ">"
>>> print(b)
<PythonStringPythonStringPythonString>
```

```
>>>
```

String length:

len() is a built-in function which returns the number of characters in a string:

```
>>> a = "Python string"
>>> len(a)
13
>>> a[13]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> a[12]
'g'
>>>
```

Traverse string with a while or for loop

- A lot of computations involve processing a string character by character at a time.
- They start at the beginning, select each character, in turn, do something to it, and continue until the end.
- This pattern of processing is called a traversal.
- One can traverse string with a while or for loop:

Code:

```
a = "STRING"
i = 0
while i < len(a):
    c = a[i]

    print(c)

    i = i + 1
```

Output:

```
>>>
S
T
R
I
N
G
>>>
```

Traversal with a for loop :

Print entire string on one line using for loop.

Code:

```
a = "Python"

i = 0

new=" "

for i in range (0,len(a)):

  b=a[i]

  # + used for concatenation

  new = new+b

  i = i + 1

  # prints each char on one line

  print(b)

  print(new)
```

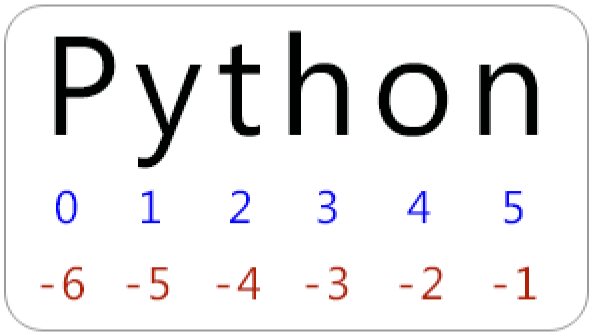
Output:

```
>>>
P
 P
y
 Py
t
 Pyt
h
 Pyth
o
 Pytho
n
 Python
>>>
```

String slices:

A segment of a string is called a slice. The "slice" syntax is used to get sub-parts of a string. The slice s[start:end] is the elements beginning at the start (p) and extending up to but not including end (q).

- The operator [p:q] returns the part of the string from the pth character to the qth character, including the first but excluding the last.
- If we omit the first index (before the colon), the slice starts at the beginning of the string.
- If you omit the second index, the slice goes to the end of the string:
- If the first index is greater than or equal to the second the result is an empty string, represented by two quotation marks.



Example-1

```
>> s = "Python"
```

- s[1:4] is 'yth' -- chars starting at index 1 and extending up to but not including index 4
- s[1:] is 'ython' -- omitting either index defaults to the start or end of the string
- s[:] is 'Python' --   of the whole thing
- s[1:100] is 'ython' -- an index that is too big is truncated down to the string length

Python uses negative numbers to give easy access to the chars at the end of the string. Negative index numbers count back from the end of the string:

- s[-1] is 'n' -- last char (1st from the end)
- s[-4] is 't' -- 4th from the end
- s[:-3] is 'pyt' -- going up to but not including the last 3 chars
- s[-3:] is 'hon' -- starting with the 3rd char from the end and extending to the end of the string.

Example-2

```
>>> a = "Python String"

>>> print (a[0:5])

Pytho

>>> print(a[6:11])

 Stri

>>> print(a[5:13])

n String

>>>
```

```
>>> a = "Python String"

>>> print(a[:8])

Python S

>>> print(a[4:])

on String
```

```
>>> print(a[6:3])


>>>
```

Example-3

```
>>> a = "PYTHON"

>>> a[3]

'H'

>>> a[0:4]

'PYTH'

>>> a[3:5]

'HO'

>>> # The first two characters

...

>>> a[:2]

'PY'

>>> # All but the first three characters

...

>>> a[3:]

'HON'

>>>
```

Search a character in a string:

Code:

```
def search(char,str):
    L=len(str)

    print(L)

    i = 0

    while i < L:

        if str[i]== char:

            return 1

            i = i + 1

        return -1


print(search("P","PYTHON"))
```

Output:

```
>>>
6
1
>>>
```

- It takes a character and finds the index where that character appears in a string.
- If the character is not found, the function returns -1.

Another example

```
def search(char,str):

    L=len(str)

    print(L)

    i = 0

    while i < L:

        if str[i]== char:

            return 1
            i = i + 1

        return -1


print(search("S","PYTHON"))
```

Output:

```
>>>
6
-1
>>>
```

012 A) Python String Formatting

String Formatting

The format() method is used to perform a string formatting operation. The string on which this method is called can contain literal text or replacement fields delimited by braces {}. Each replacement field contains either the numeric index of a positional argument or the name of a keyword argument.

Syntax:

```
str.format(*args, **kwargs)
```

Returns a   of the string where each replacement field is replaced with the string value of the corresponding argument.

Basic formatting:

Example-1:

```
>>> '{} {}'.format('Python', 'Format')

'Python Format'

>>>
```

```
>>> '{} {}'.format(10, 30)

'10 30'

>>>
```

This following statement allows re-arrange the order of display without changing the arguments.

Example-2:

```
>>> '{1} {0}'.format('Python', 'Format')

'Format Python'

>>>
```

Value conversion:

The new-style simple formatter calls by default the __format__() method of an object for its representation. If you just want to render the output of str(...) or repr(...) you can use the !s or !r conversion flags.

In %-style you usually use %s for the string representation but there is %r for a repr(...) conversion.

Setup:

```
class Data(object):

    def __str__(self):
        return 'str'

    def __repr__(self):
        return 'repr'
```

Example-1:

```
class Data(object):



    def __str__(self):
        return 'str'


    def __repr__(self):

        return 'repr'

x='{0!s} {0!r}'.format(Data())

print (x)
```

Output:

```
str repr
```

In Python 3 there exists an additional conversion flag that uses the output of repr(...) but uses ascii(...) instead.

Example-2:

```
class Data(object):


    def __repr__(self):

        return 'räpr'

x='{0!r} {0!a}'.format(Data())

print(x)
```

Output:

```
räpr r\xe4pr
```

Padding and aligning strings:

A value can be padded to a specific length. See the following examples where the value '15' is encoded as part of the format string.

Note: The padding character can be spaces or a specified character.

Example:

Align right:

```
>>> '{:>15}'.format('Python')
'         Python'
>>>
```

Align left:

```
>>> '{:15}'.format('Python')
'Python         '
>>>
```

By argument:

In the previous example, the value '15' is encoded as part of the format string. It is also possible to supply such values as an argument.

Example:

```
>>> '{:<{}s}'.format('Python', 15)
'Python         '
>>>
```

In the following example we have used '*' as a padding character.

Example:

```
>>> '{:*<15}'.format('Python')
'Python*********'
>>>
```

Align center:

Example:

```
>>> '{:^16}'.format('Python')
'     Python     '
>>>
```

Truncating long strings:

In the following example, we have truncated ten characters from the left side of a specified string.

Example:

```
>>> '{:.10}'.format('Python Tutorial')
'Python Tut'
>>>
```

By argument:

Example:

```
>>> '{:.{}}'.format('Python Tutorial', 10)
'Python Tut'
>>>
```

Combining truncating and padding

In the following example, we have combined truncating and padding.

Example:

```
>>> '{:10.10}'.format('Python')
'Python    '
>>>
```

Numbers:

Integers:

```
>>> '{:d}'.format(24)

'24'

>>>
```

Floats:

```
>>> '{:f}'.format(5.12345678123)

'5.123457'

>>>
```

Padding numbers:

Similar to strings numbers.

Example-1:

```
>>> '{:5d}'.format(24)

'   24'

>>>
```

The padding value represents the length of the complete output for floating points. In the following example '{:05.2f}' will display the float using five characters with two digits after the decimal point.

Example-2:

```
>>> '{:05.2f}'.format(5.12345678123)

'05.12'

>>>
```

Signed numbers:

By default only negative numbers are prefixed with a sign, but you can display numbers prefixed with the positive sign also.

Example-1:

```
>>> '{:+d}'.format(24)

'+24'

>>>
```

You can use a space character to indicate that negative numbers (should be prefixed with a minus symbol) and a leading space should be used for positive numbers.

Example-2:

```
>>> '{: d}'.format((- 24))

'-24'

>>>
```

Example-3:

```
>>> '{: d}'.format(24)

' 24'

>>>
```

You can control the position of the sign symbol relative to the padding.

Example-4:

```
>>> '{:=6d}'.format((- 24))

'-   24'

>>>
```

Named placeholders:

Both formatting styles support named placeholders. Here is an example:

Example-1:

```
>>> data = {'first': 'Place', 'last': 'Holder!'}

>>> '{first} {last}'.format(**data)

'Place Holder!'

>>>
```

.format() method can accept keyword arguments.

Example-2:

```
>>> '{first} {last}'.format(first='Place', last='Holder!')

'Place Holder!'

>>>
```

Datetime:

You can format and print datetime object as per your requirement.

Example:

```
>>> from datetime import datetime

>>> '{:%Y-%m-%d %H:%M}'.format(datetime(2016, 7, 26, 3, 57))

'2016-07-26 03:57'

>>>
```

Python: strftime()

strftime()

The strftime() function returns a string representing the date, controlled by an explicit format string. Format codes referring to hours, minutes or seconds will see 0 values.

Syntax:

```
date.strftime(format)
```

Complete list of formatting with examples:

```
from datetime import datetime
now = datetime.now()
print(now)
```

```
2019-08-28 14:25:35.711200
```

Code - %a : Weekday as locale’s abbreviated name.
Example:
Sun, Mon, …, Sat (en_US);
So, Mo, …, Sa (de_DE)

```
day = now.strftime("%a")
print("Weekday as locale’s abbreviated name:", day)
```

```
Weekday as locale’s abbreviated name: Wed
```

Code - %A : Weekday as locale’s full name
Example:
Sunday, Monday, …, Saturday (en_US);
Sonntag, Montag, …, Samstag (de_DE)

```
day = now.strftime("%A")
print("Weekday as locale’s full name:", day)
```

```
Weekday as locale’s full name: Wednesday
```

Code - %w: Weekday as a decimal number, where 0 is Sunday and 6 is Saturday.
Example:
0, 1, …, 6

```
day = now.strftime("%w")
print("Weekday as a decimal number:", day)
```

```
Weekday as a decimal number: 3
```
Code - %d: Day of the month as a zero-padded decimal number.
Example:
01, 02, …, 31

```
day = now.strftime("%d")
print("Day of the month as a zero-padded decimal number:", day)
```

```
Day of the month as a zero-padded decimal number: 28
```
Code - %b: Month as locale’s abbreviated name.
Example:
Jan, Feb, …, Dec (en_US);
Jan, Feb, …, Dez (de_DE)

```
day = now.strftime("%b")
print("Month as locale's abbreviated name:", day)
```

```
Month as locale's abbreviated name: Aug
```
Code - %B: Month as locale’s full name.
Example:
January, February, …, December (en_US);
Januar, Februar, …, Dezember (de_DE)

```
day = now.strftime("%B")
print("Month as locale's full Name:", day)
```

```
Month as locale's full Name: August
```
Code - %m: Month as a zero-padded decimal number.
Example:
01, 02, …, 12

```
day = now.strftime("%m")
print("Month as a zero-padded decimal number:", day)
```

```
Month as a zero-padded decimal number: 08
```
Code - %y: Year without century as a zero-padded decimal number.
Example:
00, 01, …, 99

```
day = now.strftime("%y")
print("Year without century as a zero-padded decimal number:", day)
```

```
Year without century as a zero-padded decimal number: 19
```
Code - %Y: Year with century as a decimal number.
Example:
0001, 0002, …, 2013, 2014, …, 9998, 9999

```
day = now.strftime("%Y")
print("Year with century as a decimal number:", day)
```

```
Year with century as a decimal number: 2019
```

Code - %H: Hour (24-hour clock) as a zero-padded decimal number.
Example:
00, 01, …, 23

```
day = now.strftime("%H")
print("Hour as a zero-padded decimal number:", day)
```

```
Hour as a zero-padded decimal number: 14
```

Code - %I: Hour (12-hour clock) as a zero-padded decimal number.
Example:
01, 02, …, 12

```
day = now.strftime("%I")
print("Hour as a zero-paddes decimal number:", day)
```

```
Hour as a zero-paddes decimal number: 02
```

Code - %p: Locale’s equivalent of either AM or PM.
Example:
AM, PM (en_US);
am, pm (de_DE)

```
day = now.strftime("%p")
print("Locale's equivalent of either AM Or PM:", day)
```

```
Locale's equivalent of either AM Or PM: PM
```

Code - %M: Minute as a zero-padded decimal number.
Example:
00, 01, …, 59

```
day = now.strftime("%M")
print("Minute as a zero-padded decimal number:", day)
```

```
Minute as a zero-padded decimal number: 25
```

Code - %S: Second as a zero-padded decimal number.
Example:
00, 01, …, 59

```
day = now.strftime("%S")
print("Second as a zero-padded decimal number:", day)
```

```
Second as a zero-padded decimal number: 35
```

Code - %f: Microsecond as a decimal number, zero-padded on the left.
Example:
000000, 000001, …, 999999

```
day = now.strftime("%f")
print("Microsecond as a decimal number, zero-padded on the left:", day)
```

```
Microsecond as a decimal number, zero-padded on the left: 711200
```

Code - %z: UTC offset in the form ±HHMM[SS[.ffffff]] (empty string if the object is naive).
Example:
(empty), +0000, -0400, +1030, +063415, -030712.345216

```
import time
from datetime import time, tzinfo, timedelta
class TZ1(tzinfo):
   def utcoffset(self, dt):
      return timedelta(hours=1)
   def dst(self, dt):
      return timedelta(0)
   def tzname(self,dt):
      return "+01:00"
   def  __repr__(self):
      return f"{self.__class__.__name__}()"
t = time(10, 8, 55, tzinfo=TZ1())
day = t.strftime("%z")
print("UTC offset in the form ±HHMM[SS[.ffffff]]:", day)
```

```
UTC offset in the form ±HHMM[SS[.ffffff]]: +0100
```

Code - %Z: Time zone name (empty string if the object is naive).
Example:
(empty), UTC, EST, CST

```
import time
from datetime import time, tzinfo, timedelta
class TZ1(tzinfo):
   def utcoffset(self, dt):
      return timedelta(hours=1)
   def dst(self, dt):
      return timedelta(0)
   def tzname(self,dt):
      return "+01:00"
   def  __repr__(self):
      return f"{self.__class__.__name__}()"
t = time(10, 8, 55, tzinfo=TZ1())
day = t.strftime("%Z")
print("Time zone name:", day)
```

```
Time zone name: +01:00
```

Code - %j: Day of the year as a zero-padded decimal number.
Example:
001, 002, …, 366

```
day = now.strftime("%j")
print("Day of the year as a zero-padded decimal number:", day)
```

```
Day of the year as a zero-padded decimal number: 240
```

Code - %U: Week number of the year (Sunday as the first day of the week) as a zero padded decimal number.
All days in a new year preceding the first Sunday are considered to be in week 0.
Example:
00, 01, …, 53

```
day = now.strftime("%U")
print("Week number of the year as a zero-padded decimal number:", day)
```

```
Week number of the year as a zero-padded decimal number: 34
```

Code - %W: Week number of the year (Monday as the first day of the week) as a decimal number.
All days in a new year preceding the first Monday are considered to be in week 0.
Example:
00, 01, …, 53

```
day = now.strftime("%W")
print("Week number of the year as a zero-padded decimal number:", day)
```

```
Week number of the year as a zero-padded decimal number: 34
```

Code - %c: Locale’s appropriate date and time representation.
Example:
Tue Aug 16 21:30:00 1988 (en_US);
Di 16 Aug 21:30:00 1988 (de_DE)

```
day = now.strftime("%c")
print("Locale's appropriate date and time representation:", day)
```

```
Locale's appropriate date and time representation: Wed Aug 28 14:25:35 2019
```

Code - %x: Locale’s appropriate date representation.
Example:
08/16/88 (None);
08/16/1988 (en_US);
16.08.1988 (de_DE)

```
day = now.strftime("%x")
print("Locale's appropriate date representation:", day)
```

```
Locale's appropriate date representation: 08/28/19
```

Code - %X: Locale’s appropriate time representation.
Example:
21:30:00 (en_US);
21:30:00 (de_DE)

```
day = now.strftime("%X")
print("Locale's appropriate time representation:", day)
```

```
Locale's appropriate time representation: 14:25:35
```

Code - %%: A literal '%' character.
Example:
%

```
day = now.strftime("%%")
print("A literal '%' character:", day)
```

```
A literal '%' character: %
```

Several additional directives not required by the C89 standard are included for convenience. These parameters all correspond to ISO 8601 date values. These may not be available on all platforms when used with the strftime() method.
The ISO 8601 year and ISO 8601 week directives are not interchangeable with the year and week number directives above.
Calling strptime() with incomplete or ambiguous ISO 8601 directives will raise a ValueError.
Code - %G: ISO 8601 year with century representing the year that contains the greater part of the ISO week (%V).
Example:
0001, 0002, …, 2013, 2014, …, 9998, 9999

```
day = now.strftime("%G")
print("ISO 8601 year with century representing the year that contains the gre
ater part of the ISO week:", day)
```

```
ISO 8601 year with century representing the year that contains the greater pa
rt of the ISO week: 2019
```

Code - %u: ISO 8601 weekday as a decimal number where 1 is Monday.
Example:
1, 2, …, 7

```
from datetime import datetime
now = datetime.now()
day = now.strftime("%u")
print("ISO 8601 weekday as a decimal number where 1 is Monday:", day)
```

```
ISO 8601 weekday as a decimal number where 1 is Monday: 3
```

Code - %V: ISO 8601 week as a decimal number with Monday as the first day of the week. Week 01 is the week containing Jan 4.
Example:
01, 02, …, 53

```
day = now.strftime("%V")
print("ISO 8601 week as a decimal number with Monday as the first day of the
week.:", day)
```

```
ISO 8601 week as a decimal number with Monday as the first day of the week.:
35
```

New in version 3.6: %G, %u and %V were added.