:mod:`string` --- Common string operations

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 1); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 4)

Unknown directive type "module".

```
.. module:: string
   :synopsis: Common string operations.
```

Source code: :source:`Lib/string.py`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 7); *backlink*

Unknown interpreted text role "source".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 11)

Unknown directive type "seealso".

```
.. seealso::

   :ref:`textseq`

   :ref:`string-methods`
```

String constants

The constants defined in this module are:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 23)

Unknown directive type "data".

```
.. data:: ascii_letters

   The concatenation of the :const:`ascii_lowercase` and :const:`ascii_uppercase`
   constants described below.  This value is not locale-dependent.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 29)

Unknown directive type "data".

```
.. data:: ascii_lowercase

   The lowercase letters ``'abcdefghijklmnopqrstuvwxyz'``.  This value is not
   locale-dependent and will not change.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 35)

Unknown directive type "data".

```
.. data:: ascii_uppercase
```

```
   The uppercase letters ``'ABCDEFGHIJKLMNOPQRSTUVWXYZ'``.  This value is not
   locale-dependent and will not change.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 41)

Unknown directive type "data".

```
.. data:: digits

   The string ``'0123456789'``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 46)

Unknown directive type "data".

```
.. data:: hexdigits

   The string ``'0123456789abcdefABCDEF'``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 51)

Unknown directive type "data".

```
.. data:: octdigits

   The string ``'01234567'``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 56)

Unknown directive type "data".

```
.. data:: punctuation

   String of ASCII characters which are considered punctuation characters
   in the ``C`` locale: ``!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 61)

Unknown directive type "data".

```
.. data:: printable

   String of ASCII characters which are considered printable.  This is a
   combination of :const:`digits`, :const:`ascii_letters`, :const:`punctuation`,
   and :const:`whitespace`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 68)

Unknown directive type "data".

```
.. data:: whitespace

   A string containing all ASCII characters that are considered whitespace.
   This includes the characters space, tab, linefeed, return, formfeed, and
   vertical tab.
```

Custom String Formatting

The built-in string class provides the ability to do complex variable substitutions and value formatting via the :meth:`~str.format` method described in PEP 3101. The :class:`Formatter` class in the :mod:`string` module allows you to create and customize your own string formatting behaviors using the same implementation as the built-in :meth:`~str.format` method.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 80); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 80); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 80); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 80); *backlink*

Unknown interpreted text role "meth".

The :class:`Formatter` class has the following public methods:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 89); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 91)

Unknown directive type "method".

```
.. method:: format(format_string, /, *args, **kwargs)

   The primary API method.  It takes a format string and
   an arbitrary set of positional and keyword arguments.
   It is just a wrapper that calls :meth:`vformat`.

   .. versionchanged:: 3.7
      A format string argument is now :ref:`positional-only
      <positional-only_parameter>`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 101)

Unknown directive type "method".

```
.. method:: vformat(format_string, args, kwargs)

   This function does the actual work of formatting.  It is exposed as a
   separate function for cases where you want to pass in a predefined
   dictionary of arguments, rather than unpacking and repacking the
   dictionary as individual arguments using the ``*args`` and ``**kwargs``
   syntax.  :meth:`vformat` does the work of breaking up the format string
   into character data and replacement fields.  It calls the various
   methods described below.
```

In addition, the :class:`Formatter` defines a number of methods that are intended to be replaced by subclasses:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 111); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 114)

Unknown directive type "method".

```
.. method:: parse(format_string)

   Loop over the format_string and return an iterable of tuples
   (*literal_text*, *field_name*, *format_spec*, *conversion*).  This is used
   by :meth:`vformat` to break the string into either literal text, or
   replacement fields.

   The values in the tuple conceptually represent a span of literal text
   followed by a single replacement field.  If there is no literal text
   (which can happen if two replacement fields occur consecutively), then
   *literal_text* will be a zero-length string.  If there is no replacement
   field, then the values of *field_name*, *format_spec* and *conversion*
   will be ``None``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 128)

Unknown directive type "method".

```
.. method:: get_field(field_name, args, kwargs)

   Given *field_name* as returned by :meth:`parse` (see above), convert it to
   an object to be formatted.  Returns a tuple (obj, used_key).  The default
   version takes strings of the form defined in :pep:`3101`, such as
   "0[name]" or "label.title".  *args* and *kwargs* are as passed in to
   :meth:`vformat`.  The return value *used_key* has the same meaning as the
   *key* parameter to :meth:`get_value`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 137)

Unknown directive type "method".

```
.. method:: get_value(key, args, kwargs)

   Retrieve a given field value.  The *key* argument will be either an
   integer or a string.  If it is an integer, it represents the index of the
   positional argument in *args*; if it is a string, then it represents a
   named argument in *kwargs*.

   The *args* parameter is set to the list of positional arguments to
   :meth:`vformat`, and the *kwargs* parameter is set to the dictionary of
   keyword arguments.

   For compound field names, these functions are only called for the first
   component of the field name; subsequent components are handled through
   normal attribute and indexing operations.

   So for example, the field expression '0.name' would cause
   :meth:`get_value` to be called with a *key* argument of 0.  The ``name``
   attribute will be looked up after :meth:`get_value` returns by calling the
   built-in :func:`getattr` function.

   If the index or keyword refers to an item that does not exist, then an
   :exc:`IndexError` or :exc:`KeyError` should be raised.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 160)

Unknown directive type "method".

```
.. method:: check_unused_args(used_args, args, kwargs)

   Implement checking for unused arguments if desired.  The arguments to this
   function is the set of all argument keys that were actually referred to in
   the format string (integers for positional arguments, and strings for
   named arguments), and a reference to the *args* and *kwargs* that was
```

```
passed to vformat.  The set of unused args can be calculated from these
parameters.  :meth:`check_unused_args` is assumed to raise an exception if
the check fails.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 170)

Unknown directive type "method".

```
.. method:: format_field(value, format_spec)

   :meth:`format_field` simply calls the global :func:`format` built-in.  The
   method is provided so that subclasses can override it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 175)

Unknown directive type "method".

```
.. method:: convert_field(value, conversion)

   Converts the value (returned by :meth:`get_field`) given a conversion type
   (as in the tuple returned by the :meth:`parse` method).  The default
   version understands 's' (str), 'r' (repr) and 'a' (ascii) conversion
   types.
```

Format String Syntax

The :meth:`str.format` method and the :class:`Formatter` class share the same syntax for format strings (although in the case of :class:`Formatter`, subclasses can define their own format string syntax). The syntax is related to that of :ref:`formatted string literals <f-strings>`, but it is less sophisticated and, in particular, does not support arbitrary expressions.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 188); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 188); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 188); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 188); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 194)

Unknown directive type "index".

```
.. index::
   single: {} (curly brackets); in string formatting
   single: . (dot); in string formatting
   single: [] (square brackets); in string formatting
   single: ! (exclamation); in string formatting
   single: : (colon); in string formatting
```

Format strings contain "replacement fields" surrounded by curly braces `{}`. Anything that is not contained in braces is considered literal text, which is copied unchanged to the output. If you need to include a brace character in the literal text, it can be escaped by

doubling: `{{` and `}}`.

The grammar for a replacement field is as follows:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 208)

Unknown directive type "productionlist".

```
.. productionlist:: format-string
   replacement_field: "{" [`field_name`] ["!" `conversion`] [":" `format_spec`] "}"
   field_name: arg_name ("." `attribute_name` | "[" `element_index` "]")*
   arg_name: [`identifier` | `digit`+]
   attribute_name: `identifier`
   element_index: `digit`+ | `index_string`
   index_string: <any source character except "]"> +
   conversion: "r" | "s" | "a"
   format_spec: <described in the next section>
```

In less formal terms, the replacement field can start with a *field_name* that specifies the object whose value is to be formatted and inserted into the output instead of the replacement field. The *field_name* is optionally followed by a *conversion* field, which is preceded by an exclamation point `'!'`, and a *format_spec*, which is preceded by a colon `':'`. These specify a non-default format for the replacement value.

See also the :ref:`formatspec` section.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 225); *backlink*

Unknown interpreted text role "ref".

The *field_name* itself begins with an *arg_name* that is either a number or a keyword. If it's a number, it refers to a positional argument, and if it's a keyword, it refers to a named keyword argument. If the numerical arg_names in a format string are 0, 1, 2, ... in sequence, they can all be omitted (not just some) and the numbers 0, 1, 2, ... will be automatically inserted in that order. Because *arg_name* is not quote-delimited, it is not possible to specify arbitrary dictionary keys (e.g., the strings `'10'` or `':-]'`) within a format string. The *arg_name* can be followed by any number of index or attribute expressions. An expression of the form `'.name'` selects the named attribute using :func:`getattr`, while an expression of the form `'[index]'` does an index lookup using :func:`__getitem__`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 227); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 227); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 239)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.1
   The positional argument specifiers can be omitted for :meth:`str.format`,
   so ``'{} {}'.format(a, b)`` is equivalent to ``'{0} {1}'.format(a, b)``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 243)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.4
   The positional argument specifiers can be omitted for :class:`Formatter`.
```

Some simple format string examples:

```
"First, thou shalt count to {0}"  # References first positional argument
"Bring me a {}"                   # Implicitly references the first positional argument
"From {} to {}"                   # Same as "From {0} to {1}"
"My quest is {name}"              # References keyword argument 'name'
"Weight in tons {0.weight}"       # 'weight' attribute of first positional arg
"Units destroyed: {players[0]}"   # First element of keyword argument 'players'.
```

The *conversion* field causes a type coercion before formatting. Normally, the job of formatting a value is done by the :meth:`__format__` method of the value itself. However, in some cases it is desirable to force a type to be formatted as a string, overriding its own definition of formatting. By converting the value to a string before calling :meth:`__format__`, the normal formatting logic is bypassed.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 255); *backlink***
>
> Unknown interpreted text role "meth".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 255); *backlink***
>
> Unknown interpreted text role "meth".

Three conversion flags are currently supported: `'!s'` which calls :func:`str` on the value, `'!r'` which calls :func:`repr` and `'!a'` which calls :func:`ascii`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 262); *backlink***
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 262); *backlink***
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 262); *backlink***
>
> Unknown interpreted text role "func".

Some examples:

```
"Harold's a clever {0!s}"        # Calls str() on the argument first
"Bring out the holy {name!r}"    # Calls repr() on the argument first
"More {!a}"                      # Calls ascii() on the argument first
```

The *format_spec* field contains a specification of how the value should be presented, including such details as field width, alignment, padding, decimal precision and so on. Each value type can define its own "formatting mini-language" or interpretation of the *format_spec*.

Most built-in types support a common formatting mini-language, which is described in the next section.

A *format_spec* field can also include nested replacement fields within it. These nested replacement fields may contain a field name, conversion flag and format specification, but deeper nesting is not allowed. The replacement fields within the format_spec are substituted before the *format_spec* string is interpreted. This allows the formatting of a value to be dynamically specified.

See the :ref:`formatexamples` section for some examples.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 287); *backlink***
>
> Unknown interpreted text role "ref".

Format Specification Mini-Language

"Format specifications" are used within replacement fields contained within a format string to define how individual values are presented (see :ref:`formatstrings` and :ref:`f-strings`). They can also be passed directly to the built-in :func:`format` function. Each formattable type may define how the format specification is to be interpreted.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 295); backlink

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 295); backlink

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 295); backlink

Unknown interpreted text role "func".

Most built-in types implement the following options for format specifications, although some of the formatting options are only supported by the numeric types.

A general convention is that an empty format specification produces the same result as if you had called :func:`str` on the value. A non-empty format specification typically modifies the result.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 305); backlink

Unknown interpreted text role "func".

The general form of a *standard format specifier* is:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 311)

Unknown directive type "productionlist".

```
.. productionlist:: format-spec
   format_spec: [[`fill`]`align`][`sign`][#][0][`width`][`grouping_option`][.`precision`][`type`]
   fill: <any character>
   align: "<" | ">" | "=" | "^"
   sign: "+" | "-" | " "
   width: `digit`+
   grouping_option: "_" | ","
   precision: `digit`+
   type: "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "x" | "X" | "%"
```

If a valid *align* value is specified, it can be preceded by a *fill* character that can be any character and defaults to a space if omitted. It is not possible to use a literal curly brace ("{" or "}") as the *fill* character in a :ref:`formatted string literal <f-strings>` or when using the :meth:`str.format` method. However, it is possible to insert a curly brace with a nested replacement field. This limitation doesn't affect the :func:`format` function.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 321); backlink

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 321); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 321); backlink

Unknown interpreted text role "func".

The meaning of the various alignment options is as follows:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 332)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 364)

Unknown directive type "index".

```
.. index::
   single: < (less); in string formatting
   single: > (greater); in string formatting
   single: = (equals); in string formatting
   single: ^ (caret); in string formatting
```

Option	Meaning
'<'	Forces the field to be left-aligned within the available space (this is the default for most objects).
'>'	Forces the field to be right-aligned within the available space (this is the default for numbers).
'='	Forces the padding to be placed after the sign (if any) but before the digits. This is used for printing fields in the form '+000000120'. This alignment option is only valid for numeric types. It becomes the default for numbers when '0' immediately precedes the field width.
'^'	Forces the field to be centered within the available space.

Note that unless a minimum field width is defined, the field width will always be the same size as the data to fill it, so that the alignment option has no meaning in this case.

The *sign* option is only valid for number types, and can be one of the following:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 364)

Unknown directive type "index".

```
.. index::
   single: + (plus); in string formatting
   single: - (minus); in string formatting
   single: space; in string formatting
```

Option	Meaning
'+'	indicates that a sign should be used for both positive as well as negative numbers.
'-'	indicates that a sign should be used only for negative numbers (this is the default behavior).
space	indicates that a leading space should be used on positive numbers, and a minus sign on negative numbers.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 383)

Unknown directive type "index".

```
.. index:: single: # (hash); in string formatting
```

The '#' option causes the "alternate form" to be used for the conversion. The alternate form is defined differently for different types. This option is only valid for integer, float and complex types. For integers, when binary, octal, or hexadecimal output is used, this option adds the respective prefix '0b', '0o', '0x', or '0X' to the output value. For float and complex the alternate form causes the result of the conversion to always contain a decimal-point character, even if no digits follow it. Normally, a decimal-point character appears in the result of these conversions only if a digit follows it. In addition, for 'g' and 'G' conversions, trailing zeros are not removed from the result.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 397)

Unknown directive type "index".

```
.. index:: single: , (comma); in string formatting
```

The ',' option signals the use of a comma for a thousands separator. For a locale aware separator, use the 'n' integer presentation type instead.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 403)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.1
   Added the ``','`` option (see also :pep:`378`).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 406)

Unknown directive type "index".

```
.. index:: single: _ (underscore); in string formatting
```

The `'_'` option signals the use of an underscore for a thousands separator for floating point presentation types and for integer presentation type `'d'`. For integer presentation types `'b'`, `'o'`, `'x'`, and `'X'`, underscores will be inserted every 4 digits. For other presentation types, specifying this option is an error.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 415)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.6
   Added the ``'_'`` option (see also :pep:`515`).
```

width is a decimal integer defining the minimum total field width, including any prefixes, separators, and other formatting characters. If not specified, then the field width will be determined by the content.

When no explicit alignment is given, preceding the *width* field by a zero (`'0'`) character enables sign-aware zero-padding for numeric types. This is equivalent to a *fill* character of `'0'` with an *alignment* type of `'='`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 427)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.10
   Preceding the *width* field by ``'0'`` no longer affects the default
   alignment for strings.
```

The *precision* is a decimal integer indicating how many digits should be displayed after the decimal point for presentation types `'f'` and `'F'`, or before and after the decimal point for presentation types `'g'` or `'G'`. For string presentation types the field indicates the maximum field size - in other words, how many characters will be used from the field content. The *precision* is not allowed for integer presentation types.

Finally, the *type* determines how the data should be presented.

The available string presentation types are:

Type	Meaning
`'s'`	String format. This is the default type for strings and may be omitted.
None	The same as `'s'`.

The available integer presentation types are:

Type	Meaning
`'b'`	Binary format. Outputs the number in base 2.
`'c'`	Character. Converts the integer to the corresponding unicode character before printing.
`'d'`	Decimal Integer. Outputs the number in base 10.
`'o'`	Octal format. Outputs the number in base 8.
`'x'`	Hex format. Outputs the number in base 16, using lower-case letters for the digits above 9.
`'X'`	Hex format. Outputs the number in base 16, using upper-case letters for the digits above 9. In case `'#'` is specified, the prefix `'0x'` will be upper-cased to `'0X'` as well.
`'n'`	Number. This is the same as `'d'`, except that it uses the current locale setting to insert the appropriate number separator characters.
None	The same as `'d'`.

In addition to the above presentation types, integers can be formatted with the floating point presentation types listed below (except `'n'` and `None`). When doing so, :func:`float` is used to convert the integer to a floating point number before formatting.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 481); *backlink*

Unknown interpreted text role "func".

The available presentation types for :class:`float` and :class:`~decimal.Decimal` values are:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 486); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 486); *backlink*

Unknown interpreted text role "class".

Type	Meaning
`'e'`	Scientific notation. For a given precision `p`, formats the number in scientific notation with the letter 'e' separating the coefficient from the exponent. The coefficient has one digit before and `p` digits after the decimal point, for a total of `p + 1` significant digits. With no precision given, uses a precision of `6` digits after the decimal point for :class:`float`, and shows all coefficient digits for :class:`~decimal.Decimal`. If no digits follow the decimal point, the decimal point is also removed unless the `#` option is used. System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 493); *backlink* Unknown interpreted text role "class". System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 493); *backlink* Unknown interpreted text role "class".
`'E'`	Scientific notation. Same as `'e'` except it uses an upper case 'E' as the separator character.
`'f'`	Fixed-point notation. For a given precision `p`, formats the number as a decimal number with exactly `p` digits following the decimal point. With no precision given, uses a precision of `6` digits after the decimal point for :class:`float`, and uses a precision large enough to show all coefficient digits for :class:`~decimal.Decimal`. If no digits follow the decimal point, the decimal point is also removed unless the `#` option is used. System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 508); *backlink* Unknown interpreted text role "class". System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 508); *backlink* Unknown interpreted text role "class".
`'F'`	Fixed-point notation. Same as `'f'`, but converts `nan` to `NAN` and `inf` to `INF`.

Type	Meaning
`'g'`	General format. For a given precision `p >= 1`, this rounds the number to `p` significant digits and then formats the result in either fixed-point format or in scientific notation, depending on its magnitude. A precision of `0` is treated as equivalent to a precision of `1`. The precise rules are as follows: suppose that the result formatted with presentation type `'e'` and precision `p-1` would have exponent `exp`. Then, if `m <= exp < p`, where `m` is -4 for floats and -6 for :class:`Decimals <decimal.Decimal>`, the number is formatted with presentation type `'f'` and precision `p-1-exp`. Otherwise, the number is formatted with presentation type `'e'` and precision `p-1`. In both cases insignificant trailing zeros are removed from the significand, and the decimal point is also removed if there are no remaining digits following it, unless the `'#'` option is used. **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 528); *backlink*** Unknown interpreted text role "class". With no precision given, uses a precision of `6` significant digits for :class:`float`. For :class:`~decimal.Decimal`, the coefficient of the result is formed from the coefficient digits of the value; scientific notation is used for values smaller than `1e-6` in absolute value and values where the place value of the least significant digit is larger than 1, and fixed-point notation is used otherwise. **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 541); *backlink*** Unknown interpreted text role "class". **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 541); *backlink*** Unknown interpreted text role "class". Positive and negative infinity, positive and negative zero, and nans, are formatted as `inf`, `-inf`, `0`, `-0` and `nan` respectively, regardless of the precision.
`'G'`	General format. Same as `'g'` except switches to `'E'` if the number gets too large. The representations of infinity and NaN are uppercased, too.
`'n'`	Number. This is the same as `'g'`, except that it uses the current locale setting to insert the appropriate number separator characters.
`'%'`	Percentage. Multiplies the number by 100 and displays in fixed (`'f'`) format, followed by a percent sign.

Type	Meaning
None	For :class:`float` this is the same as `'g'`, except that when fixed-point notation is used to format the result, it always includes at least one digit past the decimal point. The precision used is as large as needed to represent the given value faithfully. System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 566); *backlink* Unknown interpreted text role "class". For :class:`~decimal.Decimal`, this is the same as either `'g'` or `'G'` depending on the value of `context.capitals` for the current decimal context. System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 572); *backlink* Unknown interpreted text role "class". The overall effect is to match the output of :func:`str` as altered by the other format modifiers. System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 576); *backlink* Unknown interpreted text role "func".

Format examples

This section contains examples of the :meth:`str.format` syntax and comparison with the old `%`-formatting.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 585); *backlink*

Unknown interpreted text role "meth".

In most of the cases the syntax is similar to the old `%`-formatting, with the addition of the `{}` and with `:` used instead of `%`. For example, `'%03.2f'` can be translated to `'{:03.2f}'`.

The new format syntax also supports new and different options, shown in the following examples.

Accessing arguments by position:

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> '{}, {}, {}'.format('a', 'b', 'c')  # 3.1+ only
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
'c, b, a'
>>> '{2}, {1}, {0}'.format(*'abc')      # unpacking argument sequence
'c, b, a'
>>> '{0}{1}{0}'.format('abra', 'cad')   # arguments' indices can be repeated
'abracadabra'
```

Accessing arguments by name:

```
>>> 'Coordinates: {latitude}, {longitude}'.format(latitude='37.24N', longitude='-115.81W')
'Coordinates: 37.24N, -115.81W'
>>> coord = {'latitude': '37.24N', 'longitude': '-115.81W'}
>>> 'Coordinates: {latitude}, {longitude}'.format(**coord)
'Coordinates: 37.24N, -115.81W'
```

Accessing arguments' attributes:

```
>>> c = 3-5j
>>> ('The complex number {0} is formed from the real part {0.real} '
...  'and the imaginary part {0.imag}.').format(c)
'The complex number (3-5j) is formed from the real part 3.0 and the imaginary part -5.0.'
>>> class Point:
...     def __init__(self, x, y):
...         self.x, self.y = x, y
```

```
...     def __str__(self):
...         return 'Point({self.x}, {self.y})'.format(self=self)
...
>>> str(Point(4, 2))
'Point(4, 2)'
```

Accessing arguments' items:

```
>>> coord = (3, 5)
>>> 'X: {0[0]};  Y: {0[1]}'.format(coord)
'X: 3;  Y: 5'
```

Replacing `%s` and `%r`:

```
>>> "repr() shows quotes: {!r}; str() doesn't: {!s}".format('test1', 'test2')
"repr() shows quotes: 'test1'; str() doesn't: test2"
```

Aligning the text and specifying a width:

```
>>> '{:<30}'.format('left aligned')
'left aligned                  '
>>> '{:>30}'.format('right aligned')
'                 right aligned'
>>> '{:^30}'.format('centered')
'           centered           '
>>> '{:*^30}'.format('centered')  # use '*' as a fill char
'***********centered***********'
```

Replacing `%+f`, `%-f`, and `% f` and specifying a sign:

```
>>> '{:+f}; {:+f}'.format(3.14, -3.14)  # show it always
'+3.140000; -3.140000'
>>> '{: f}; {: f}'.format(3.14, -3.14)  # show a space for positive numbers
' 3.140000; -3.140000'
>>> '{:-f}; {:-f}'.format(3.14, -3.14)  # show only the minus -- same as '{:f}; {:f}'
'3.140000; -3.140000'
```

Replacing `%x` and `%o` and converting the value to different bases:

```
>>> # format also supports binary numbers
>>> "int: {0:d};  hex: {0:x};  oct: {0:o};  bin: {0:b}".format(42)
'int: 42;  hex: 2a;  oct: 52;  bin: 101010'
>>> # with 0x, 0o, or 0b as prefix:
>>> "int: {0:d};  hex: {0:#x};  oct: {0:#o};  bin: {0:#b}".format(42)
'int: 42;  hex: 0x2a;  oct: 0o52;  bin: 0b101010'
```

Using the comma as a thousands separator:

```
>>> '{:,}'.format(1234567890)
'1,234,567,890'
```

Expressing a percentage:

```
>>> points = 19
>>> total = 22
>>> 'Correct answers: {:.2%}'.format(points/total)
'Correct answers: 86.36%'
```

Using type-specific formatting:

```
>>> import datetime
>>> d = datetime.datetime(2010, 7, 4, 12, 15, 58)
>>> '{:%Y-%m-%d %H:%M:%S}'.format(d)
'2010-07-04 12:15:58'
```

Nesting arguments and more complex examples:

```
>>> for align, text in zip('<^>', ['left', 'center', 'right']):
...     '{0:{fill}{align}16}'.format(text, fill=align, align=align)
...
'left<<<<<<<<<<<<'
'^^^^^center^^^^^'
'>>>>>>>>>>>right'
>>>
>>> octets = [192, 168, 0, 1]
>>> '{:02X}{:02X}{:02X}{:02X}'.format(*octets)
'C0A80001'
>>> int(_, 16)
3232235521
>>>
>>> width = 5
>>> for num in range(5,12): #doctest: +NORMALIZE_WHITESPACE
...     for base in 'dXob':
```

```
...         print('{0:{width}{base}}'.format(num, base=base, width=width), end=' ')
...     print()
...
    5     5     5   101
    6     6     6   110
    7     7     7   111
    8     8    10  1000
    9     9    11  1001
   10     A    12  1010
   11     B    13  1011
```

Template strings

Template strings provide simpler string substitutions as described in PEP 292. A primary use case for template strings is for internationalization (i18n) since in that context, the simpler syntax and functionality makes it easier to translate than other built-in string formatting facilities in Python. As an example of a library built on template strings for i18n, see the flufl.i18n package.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 734)

Unknown directive type "index".

```
.. index:: single: $ (dollar); in template strings
```

Template strings support `$`-based substitutions, using the following rules:

- `$$` is an escape; it is replaced with a single `$`.
- `$identifier` names a substitution placeholder matching a mapping key of `"identifier"`. By default, `"identifier"` is restricted to any case-insensitive ASCII alphanumeric string (including underscores) that starts with an underscore or ASCII letter. The first non-identifier character after the `$` character terminates this placeholder specification.
- `${identifier}` is equivalent to `$identifier`. It is required when valid identifier characters follow the placeholder but are not part of the placeholder, such as `"${noun}ification"`.

Any other appearance of `$` in the string will result in a :exc:`ValueError` being raised.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 751); *backlink*

Unknown interpreted text role "exc".

The :mod:`string` module provides a :class:`Template` class that implements these rules. The methods of :class:`Template` are:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 754); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 754); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 754); *backlink*

Unknown interpreted text role "class".

The constructor takes a single argument which is the template string.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 763)

Unknown directive type "method".

```
.. method:: substitute(mapping={}, /, **kwds)

   Performs the template substitution, returning a new string.  *mapping* is
   any dictionary-like object with keys that match the placeholders in the
   template.  Alternatively, you can provide keyword arguments, where the
   keywords are the placeholders.  When both *mapping* and *kwds* are given
```

```
and there are duplicates, the placeholders from *kwds* take precedence.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 772)

Unknown directive type "method".

```
.. method:: safe_substitute(mapping={}, /, **kwds)

   Like :meth:`substitute`, except that if placeholders are missing from
   *mapping* and *kwds*, instead of raising a :exc:`KeyError` exception, the
   original placeholder will appear in the resulting string intact.  Also,
   unlike with :meth:`substitute`, any other appearances of the ``$`` will
   simply return ``$`` instead of raising :exc:`ValueError`.

   While other exceptions may still occur, this method is called "safe"
   because it always tries to return a usable string instead of
   raising an exception.  In another sense, :meth:`safe_substitute` may be
   anything other than safe, since it will silently ignore malformed
   templates containing dangling delimiters, unmatched braces, or
   placeholders that are not valid Python identifiers.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 788)

Unknown directive type "method".

```
.. method:: is_valid()

   Returns false if the template has invalid placeholders that will cause
   :meth:`substitute` to raise :exc:`ValueError`.

   .. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 796)

Unknown directive type "method".

```
.. method:: get_identifiers()

   Returns a list of the valid identifiers in the template, in the order
   they first appear, ignoring any invalid identifiers.

   .. versionadded:: 3.11
```

:class:`Template` instances also provide one public data attribute:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 803); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 805)

Unknown directive type "attribute".

```
.. attribute:: template

   This is the object passed to the constructor's *template* argument.  In
   general, you shouldn't change it, but read-only access is not enforced.
```

Here is an example of how to use a Template:

```
>>> from string import Template
>>> s = Template('$who likes $what')
```

```
>>> s.substitute(who='tim', what='kung pao')
'tim likes kung pao'
>>> d = dict(who='tim')
>>> Template('Give $who $100').substitute(d)
Traceback (most recent call last):
...
ValueError: Invalid placeholder in string: line 1, col 11
>>> Template('$who likes $what').substitute(d)
Traceback (most recent call last):
...
KeyError: 'what'
>>> Template('$who likes $what').safe_substitute(d)
'tim likes $what'
```

Advanced usage: you can derive subclasses of :class:`Template` to customize the placeholder syntax, delimiter character, or the entire regular expression used to parse template strings. To do this, you can override these class attributes:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 828); *backlink***
>
> Unknown interpreted text role "class".

- *delimiter* -- This is the literal string describing a placeholder introducing delimiter. The default value is `$`. Note that this should *not* be a regular expression, as the implementation will call :meth:`re.escape` on this string as needed. Note further that you cannot change the delimiter after class creation (i.e. a different delimiter must be set in the subclass's class namespace).

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 833); *backlink***
  >
  > Unknown interpreted text role "meth".

- *idpattern* -- This is the regular expression describing the pattern for non-braced placeholders. The default value is the regular expression `(?a:[_a-z][_a-z0-9]*)`. If this is given and *braceidpattern* is `None` this pattern will also apply to braced placeholders.

  > **Note**
  >
  > Since default *flags* is `re.IGNORECASE`, pattern `[a-z]` can match with some non-ASCII characters. That's why we use the local `a` flag here.

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 851)**
  >
  > Unknown directive type "versionchanged".
  >
  > ```
  > .. versionchanged:: 3.7
  >    *braceidpattern* can be used to define separate patterns used inside and
  >    outside the braces.
  > ```

- *braceidpattern* -- This is like *idpattern* but describes the pattern for braced placeholders. Defaults to `None` which means to fall back to *idpattern* (i.e. the same pattern is used both inside and outside braces). If given, this allows you to define different patterns for braced and unbraced placeholders.

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 861)**
  >
  > Unknown directive type "versionadded".
  >
  > ```
  > .. versionadded:: 3.7
  > ```

- *flags* -- The regular expression flags that will be applied when compiling the regular expression used for recognizing substitutions. The default value is `re.IGNORECASE`. Note that `re.VERBOSE` will always be added to the flags, so custom *idpattern*s must follow conventions for verbose regular expressions.

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 869)**
  >
  > Unknown directive type "versionadded".

```
.. versionadded:: 3.2
```

Alternatively, you can provide the entire regular expression pattern by overriding the class attribute *pattern*. If you do this, the value must be a regular expression object with four named capturing groups. The capturing groups correspond to the rules given above, along with the invalid placeholder rule:

- *escaped* -- This group matches the escape sequence, e.g. `$$`, in the default pattern.
- *named* -- This group matches the unbraced placeholder name; it should not include the delimiter in capturing group.
- *braced* -- This group matches the brace enclosed placeholder name; it should not include either the delimiter or braces in the capturing group.
- *invalid* -- This group matches any other delimiter pattern (usually a single delimiter), and it should appear last in the regular expression.

The methods on this class will raise :exc:`ValueError` if the pattern matches the template without one of these named groups matching.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 889); *backlink*

Unknown interpreted text role "exc".

Helper functions

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]string.rst, line 896)

Unknown directive type "function".

```
.. function:: capwords(s, sep=None)

   Split the argument into words using :meth:`str.split`, capitalize each word
   using :meth:`str.capitalize`, and join the capitalized words using
   :meth:`str.join`.  If the optional second argument *sep* is absent
   or ``None``, runs of whitespace characters are replaced by a single space
   and leading and trailing whitespace are removed, otherwise *sep* is used to
   split and join the words.
```