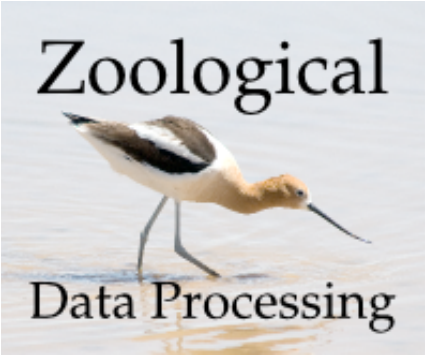
Next / Previous / Contents / Shipman's homepage

# Python 2.7 quick reference




## 9.4. The string `.format()` method

The `.format()` method of the `str` type is an extremely convenient way to format text exactly the way you want it.

### Note

This method was added in Python 2.6.

Quite often, we want to embed data values in some explanatory text. For example, if we are displaying the number of nematodes in a hectare, it is a lot more meaningful to display it as `"There were 37.9 nematodes per hectare"` than just `"37.9"`. So what we need is a way to mix constant text like `"nematodes per hectare"` with values from elsewhere in your program.

Here is the general form:

```
template.format(p0, p1, ..., k0=v0, k1=v1, ...)
```

The *`template`* is a string containing a mixture of one or more *format codes* embedded in constant text. The `format` method uses its arguments to substitute an appropriate value for each format code in the template.

The arguments to the `.format()` method are of two types. The list starts with zero or more positional arguments $p_i$, followed by zero or more keyword arguments of the form $k_i=v_i$, where each $k_i$ is a name with an associated value $v_i$.

Just to give you the general flavor of how this works, here's a simple conversational example. In this example, the format code "`{0}`" is replaced by the first positional argument (49), and "`{1}`" is replaced by the second positional argument, the string `"okra"`.

```
>>> "We have {0} hectares planted to {1}.".format(49, "okra")
'We have 49 hectares planted to okra.'
>>>
```

In the next example, we supply the values using keyword arguments. The arguments may be supplied in any order. The keyword names must be valid Python names (see Section 5, "Names and keywords").

```
>>> "{monster} has now eaten {city}".format(
```

```
...     city='Tokyo', monster='Mothra')
'Mothra has now eaten Tokyo'
```

You may mix references to positional and keyword arguments:

```
>>> "The {structure} sank {0} times in {1} years.".format(
...   3, 2, structure='castle')
'The castle sank 3 times in 2 years.'
```

If you need to include actual "{" and "}" characters in the result, double them, like this:

```
>>> "There are {0} members in set {{a}}.".format(15)
'There are 15 members in set {a}.'
```

### 9.4.1. General form of a format code

Here is the general form of a format code, where optional parts in `[brackets]`, and actual characters are in `"double quotes"`:

```
"{" [name] ["!" conversion] [":" spec] "}"
```

- For the *name* portion, see Section 9.4.2, "The *name* part".
- For the *conversion* part, see Section 9.4.3, "The *conversion* part".
- For the spec part, see Section 9.4.4, "The *spec* part".

### 9.4.2. The *name* part

The *name* part of a format code specifies the source of the value to be formatted here. Numbers refer to positional arguments passed to the `.format()` method, starting at 0 for the first argument. You may also use any Python name to refer to one of the keyword arguments.

- If the associated argument is an iterable, you may append an expression of this form to retrieve one of its elements:

  ```
  "[" index "]"
  ```

  For example:

  ```
  >>> signal=['red', 'yellow', 'green']
  >>> signal[2]
  'green'
  >>> "The light is {0[2]}!".format(signal)
  'The light is green!'
  ```

- If the associated argument has attributes, you can append an expression of this form to refer to that attribute:

  ```
  "."name
  ```

  For example:

```
>>> import string
>>> string.digits
'0123456789'
>>> "Our digits are '{s.digits}'.".format(s=string)
"Our digits are '0123456789'."
```

In general, you can use any combination of these features. For example:

```
>>> "The sixth digit is '{s.digits[5]}'".format(s=string)
"The sixth digit is '5'"
```

Starting with Python 2.7, you may omit all of the numbers that refer to positional arguments, and they will be used in the sequence they occur. For example:

```
>>> "The date is {}-{}-{}.".format(2012, 5, 1)
'The date is 2012-5-1.'
```

If you use this convention, you must omit all those numbers. You can, however, omit all the numbers and still use the keyword names feature:

```
>>> "Can I have {} pounds to {excuse}?".format(
...   50, excuse='mend the shed')
'Can I have 50 pounds to mend the shed?'
```

### 9.4.3. The *conversion* part

Following the *name* part of a format code, you can use one of these two forms to force the value to be converted by a standard function:

| | |
|---|---|
| !s | str() |
| !r | repr() |

Here's an example:

```
>>> "{}".format('Don\'t')
"Don't"
>>> "{!r}".format('Don\'t')
'"Don\'t"'
```

### 9.4.4. The *spec* part

After the *name* and *conversion* parts of a format code, you may use a colon (“:”) and a format specifier to supply more details about how to format the related value.

Here is the general form of a format specifier.

```
":" [[fill] align] [sign] ["#"] ["0"] [width] [","] ["." prec] [type]
```

***fill***

You may specify any fill character except "}". This character is used to pad a short value to the specified length. It may be specified only in combination with an *align* character.

***align***

Specifies how to align values that are not long enough to occupy the specified length. There are four values:

| | |
|---|---|
| < | Left-justify the value. This is the default alignment for string values. |
| > | Right-justify the value. This is the default alignment for numbers. |
| ^ | Center the value. |
| = | For numbers using a *sign* specifier, add the padding between the sign and the rest of the value. |

Here are some examples of the use of *fill* and *align*.

```
>>> "{:>8}".format(13)
'      13'
>>> "{:>8}".format('abc')
'     abc'
>>> "{:*>8}".format('abc')
'*****abc'
>>> "{:*<8}".format('abc')
'abc*****'
>>> "{:>5d}".format(14)
'   14'
>>> "{:#>5d}".format(14)
'###14'
>>> "{:<6}".format('Git')
'Git   '
>>> "{:*<6}".format('Git')
'Git***'
>>> "{:=^8}".format('Git')
'==Git==='
>>> "{:*=-9d}".format(-3)
'-*******3'
```

***sign***

This option controls whether an arithmetic sign is displayed. There are three possible values:

| | |
|---|---|
| + | Always display a sign: + for positive, - for negative. |
| - | Display - only for negative values. |
| (one space) | Display one space for positive values, - for negative. |

Here are some examples of use of the sign options.

```
>>> '{} {}'.format(17, -17)
'17 -17'
>>> '{:5} {:5}'.format(17, -17)
'   17   -17'
>>> '{:<5} {:<5}'.format(17, -17)
'17    -17  '
```

```
>>> '{:@<5} {:@<5}'.format(17, -17)
'17@@@ -17@@'
>>> '{:@>5} {:@>5}'.format(17, -17)
'@@@17 @@-17'
>>> '{:@^5} {:@^5}'.format(17, -17)
'@17@@ @-17@'
>>> '{:@^+5} {:@^+5}'.format(17, -17)
'@+17@ @-17@'
>>> '{:@^-5} {:@^-5}'.format(17, -17)
'@17@@ @-17@'
>>> '{:@^ 5} {:@^ 5}'.format(17, -17)
'@ 17@ @-17@'
```

**"#"**

This option selects the "alternate form" of output for some types.

- When formatting integers as binary, octal, or hexadecimal, the alternate form adds "`0b`", "`0o`", or "`0x`" before the value, to show the radix explicitly.

```
>>> "{:4x}".format(255)
'  ff'
>>> "{:#4x}".format(255)
'0xff'
>>>  "{:9b}".format(62)
'   111110'
>>> "{:#9b}".format(62)
' 0b111110'
>>> "{:<#9b}".format(62)
'0b111110 '
```

- When formatting `float`, `complex`, or `Decimal` values, the "#" option forces the result to contain a decimal point, even if it is a whole number.

```
>>>
"{:5.0f}".format(36)
'   36'
>>> "{:#5.0f}".format(36)
'  36.'
>>> from decimal import Decimal
>>> w=Decimal(36)
>>> "{:g}".format(w)
'36'
>>> "{:#g}".format(w)
'36.'
```

**"0"**

To fill the field with left zeroes, place a "`0`" at this position in your format code.

```
>>> "{:5d}".format(36)
'   36'
>>> "{:05d}".format(36)
'00036'
>>> "{:021.15}".format(1.0/7.0)
'00000.142857142857143'
```

***width***

Place a number at this position to specify the total width of the displayed value.

```
>>> "Beware the {}!".format('Penguin')
'Beware the Penguin!'
>>> "Beware the {:11}!".format('Penguin')
'Beware the Penguin    !'
>>> "Beware the {:>11}!".format('Penguin')
'Beware the     Penguin!'
```

`","`

Place a comma at this position in your format code to display commas between groups of three digits in whole numbers.

**Note**

This feature was added in Python 2.7.

```
>>>  "{:,d}".format(12345678901234)
'12,345,678,901,234'
>>> "{:,f}".format(1234567890123.456789)
'1,234,567,890,123.456787'
>>> "{:25,f}".format(98765432.10987)
'        98,765,432.109870'
```

`"."` ***precision***

Use this part to specify the number of digits after the decimal point.

```
>>> from math import pi
>>> "{}".format(pi)
'3.141592653589793'
>>> "{:.3}".format(pi)
'3.14'
>>> "{:25,.3f}".format(1234567890123.456789)
'    1,234,567,890,123.457'
```

***type***

This code specifies the general type of format used. The default is to convert the value of a string as if using the `str()` function. Refer to the table below for allowed values.

| | |
|---|---|
| `b` | Format an integer in binary. |
| `c` | Given a number, display the character that has that code. |
| `d` | Display a number in decimal (base 10). |
| `e` | Display a `float` value using the exponential format. |
| `E` | Same as `e`, but use a capital "`E`" in the exponent. |
| `f` | Format a number in fixed-point form. |
| `g` | General numeric format: use either `f` or `g`, whichever is appropriate. |
| `G` | Same as "g", but uses a capital "`E`" in the exponential form. |
| `n` | For formatting numbers, this format uses the current local setting to insert separator characters. For example, a number that Americans would show as "`1,234.56`", Europeans would show it |

| | |
|---|---|
| | as “1.234,56”. |
| o | Display an integer in octal format. |
| x | Display an integer in hexadecimal (base 16). Digits greater than 9 are displayed as lowercase characters. |
| X | Display an integer in hexadecimal (base 16). Digits greater than 9 are displayed as uppercase characters. |
| % | Display a number as a percentage: its value is multiplied by 100, followed by a “%” character. |

Examples:

```
>>> "{:b}".format(9)
'1001'
>>> "{:08b}".format(9)
'00001001'
>>> "{:c}".format(97)
'a'
>>> "{:d}".format(0xff)
'255'
>>> from math import pi
>>> "{:e}".format(pi*1e10)
'3.141593e+10'
>>> "{:E}".format(pi*1e10)
'3.141593E+10'
>>> "{:f}".format(pi)
'3.141593'
>>> "{:g}".format(pi)
'3.14159'
>>> "{:g}".format(pi*1e37)
'3.14159e+37'
>>> "{:G}".format(pi*1e37)
'3.14159E+37'
>>> "{:o}".format(255)
'377'
>>> "{:#o}".format(255)
'0o377'
>>> "{:x}".format(105199)
'19aef'
>>> "{:X}".format(105199)
'19AEF'
>>> "{:<#9X}".format(105199)
'0X19AEF  '
>>> "{:%}".format(0.6789)
'67.890000%'
>>> "{:15.3%}".format(0.6789)
'        67.890%'
```

### 9.4.5. Formatting a field of variable length

Sometimes you need to format a field using a length that is available only once the program is running. To do this, you can use a number or name in `{braces}` *inside* a format code at the *`width`* position. This item then refers to either a positional or keyword argument to the `.format()` method as usual.

Here's an example. Suppose you want to format a number `n` using `d` digits. Here are examples showing this with and without left-zero fill:

```
>>> n = 42
>>> d = 8
>>> "{0:{1}d}".format(42, 8)
'      42'
>>> "{0:0{1}d}".format(42, 8)
'00000042'
>>>
```

You can, of course, also use keyword arguments to specify the field width. This trick also works for variable precision.

```
"{count:0{width}d}".format(width=8, count=42)
'00000042'
>>>
```

The same technique applies to substituting any of the pieces of a format code.

```
>>> "{:&<14,d}".format(123456)
'123,456&&&&&&'
>>> "{1:{0}{2}{3},{4}}".format('&', 123456, '<', 14, 'd')
'123,456&&&&&&'
>>> "{:@^14,d}".format(1234567)
'@@1,234,567@@@'
>>> "{n:{fil}{al}{w},{kind}}".format(
...     kind='d', w=14, al='^', fil='@', n=1234567)
'@@1,234,567@@@'
```

---

**Next:** 9.5. The older string format operator
**Contents:** Python 2.7 quick reference
**Previous:** 9.3. Methods on `str` values
**Zoological Data Processing homepage**
**Shipman's Home Sweet Homepage**

---

*John W. Shipman*
*Comments welcome: john@nmt.edu*
Last updated: 2015-03-03 15:23
URL: `http://infohost.nmt.edu/~shipman/doc/python27/web/new-str-format.html`