

Python ≥ 2.6: formatação com `format()` e `str.format()`

A especificação PEP 3101 traz uma sintaxe nova para formatação de valores em strings, que deve ser usada preferencialmente em vez da formatação com operador `%` (que poderá ser abandonada). A nova notação é usada principalmente em dois contextos:

`str.format(*args, **kwargs)` Método aplicado a string contendo marcas de substituição `{s!c:f}` onde:

- s** seleciona o argumento a formatar,
- c** especifica uma conversão e
- f** especifica o formato de apresentação

`format(valor, f)` função *built-in* onde

- f** especifica o formato de apresentação

Nos dois casos o método `o.__format__(f)` é invocado em cada objeto a exibir. Classes podem implementar este método para criar códigos de formatação customizados. A nova classe **`string.Formatter`** facilita este processo.

```
>>> import math
>>> format(math.pi, '6.3f')
' 3.142'
>>> fmt = '{0} com 4 casas: {0:.4f}'
>>> fmt.format(math.pi)
'3.14159265359 com 4 casas: 3.1416'
>>> fmt2 = '{0} com {n:02} casas: {0:.{n}f}'
>>> fmt2.format(math.pi, n=5)
'3.14159265359 com 05 casas: 3.14159'
>>> print u'{} ≈ {}'.format(u'π', math.pi)
π ≈ 3.14159265359
```

Marcas de substituição {...}



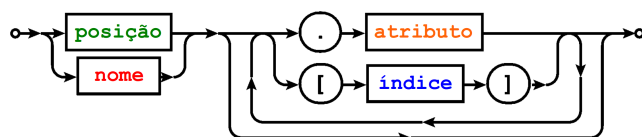
argumento Indica qual argumento do método `format` será apresentado no lugar desta marca de substituição. Veja a seção **Seletor do argumento** abaixo.

conversão Use para forçar a conversão do argumento para string usando as funções `str` ou `repr`. Por padrão, a conversão é feita pela invocação do método `obj.__format__(fmt)`, onde `obj` é o argumento e `fmt` é a **Especificação do formato**. Veja o exemplo da classe **`Spam`** ao lado.

formato Especificação do formato de apresentação. Veja a seção **Especificação do formato** no verso.

```
>>> '{0} {1} {2}'.format(2, 3, 5)
'2 3 5'
>>> '{} {} {}'.format(2, 3, 5) # Python ≥ 2.7
'2 3 5'
>>> '{0.real} {0.imag}'.format(3j+4)
'4.0 3.0'
>>> '{0.real:f} {0.imag:f}'.format(3j+4)
'4.000000 3.000000'
>>> d = {'BRL':0.5457, 'EUR':1.3496}
>>> 'Euro:{0[EUR]}, Real:{0[BRL]}'.format(d)
'Euro:1.3496, Real:0.5457'
>>> 'Euro:{EUR}, Real:{BRL}'.format(**d)
'Euro:1.3496, Real:0.5457'
>>> from datetime import date
>>> dts = (date(2011,9,3), date(2011,9,7))
>>> 'de {0[0].day} a {0[1].day}'.format(dts)
'de 3 a 7'
>>> 'de {:.day} a {:.day}'.format(*dts) # ≥ 2.7
'de 3 a 7'
>>> class Spam(object):
...     def __str__(self):
...         return 'Spam!!!'
...     def __format__(self, fmt):
...         return 'Spam'.replace(fmt, fmt*3)
>>> s = Spam()
>>> '{0!s}, {0!r}'.format(s) #doctest:
+ELLIPSIS
'Spam!!!, <__main__.Spam object at ...>'
>>> '{0}, {0:a}, {0:m}'.format(s)
'Spam, Spaaam, Spamm'
```

Seletor do argumento {x...}



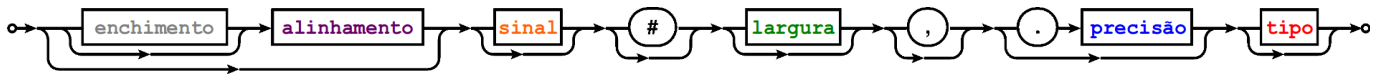
posição Inteiro para selecionar um argumento posicional passado para `str.format(*args)`. A posição e o nome podem ser omitidos para exibir os argumentos posicionais em ordem.

nome Identificador de um argumento nomeado passado para `str.format(**kwargs)`.

atributo Identificador de um atributo do argumento.

índice Índice inteiro ou chave de dicionário para

Especificação do formato



alinhamento Um dos sinais <, ^, > ou =, indicando:

< alinhamento à esquerda

^ centralizado

> à direita

= à direita com preenchimento após o sinal

O sinal pode ser precedido de um caractere qualquer (exceto { ou }) a ser usado em vez do espaço para preencher o campo se **largura** for definida.

sinal Os caracteres +, - ou _ (um espaço em branco).

+ sempre exibir sinal + ou - à esquerda

- exibir apenas sinal - nos números de negativos

_ (espaço em branco) exibir sinal - à esquerda de números negativos e branco à esquerda dos positivos.

Use para exibir **0b**, **0o** ou **0x** à esquerda do número nas apresentações de **tipo** binário, octal ou hexadecimal.

largura Número de caracteres da largura total mínima do campo. O conteúdo não é truncado se exceder essa largura. Se o conteúdo for menor, haverá preenchimento conforme o **alinhamento** definido. Se a largura começar com um 0 (zero), o campo será preenchido com zeros à esquerda (igual a **alinhamento 0=**)

, Exibir , (vírgula) como separador de milhares. Para outros separadores de milhares, use o **tipo n**.

precisão Um . (ponto) seguido de um inteiro cuja função depende do **tipo** especificado.

No **tipo s**, precisão é o máximo de caracteres

No **tipo f**, é o número de dígitos após o ponto

No **tipo g** ou **n**, é o total de dígitos significativos

Não pode ser usado com os tipos **b**, **c**, **d** ou **o**

tipo Um dos caracteres abaixo. Se omitido, vale o assinalado com * para cada tipo (ex. **d** para **int**):

s **str/unicode** *

b **int** como binário

c **int** como caractere Unicode correspondente

d **int** como decimal *

o **int** como octal

x X **int** como hexadecimal: **x** caixa baixa, **X** alta

e E **float** em notação exponencial:

e caixa baixa, **E** alta

f F **float** sem usar notação exponencial

g G **float** como nos tipos **e E** ou **f F**, conforme a **precisão** e o expoente *

n **float** como no tipo **g**, usando separadores decimal e de milhares conforme o locale ativo

% **float** como porcentagem, usando formato do tipo **f**, com o valor ×100, seguido do sinal %

```
>>> format('Fotografia', '<16')
'Fotografia.....'
>>> format('Fotografia', '>16')
'.....Fotografia'
>>> format('Fotografia', '^16')
'...Fotografia...'
>>> format(math.pi, '_>+8.3f')
'__+3.142'
>>> format(123, '0= 6x')
' 0007b'
>>> format(123, '0=+6x')
'+0007b'
>>> format(123, '#06x')
'0x007b'
>>> '{0:f} {0:e}'.format(2**32)
'4294967296.000000 4.294967e+09'
>>> '{0:{1}} {0:{2}}'.format(2**32, 'f', 'e')
'4294967296.000000 4.294967e+09'
>>> format(12345678.9876, '18.10n')
'      12345678.99'
>>> from locale import setlocale, LC_NUMERIC
>>> setlocale(LC_NUMERIC, 'de_DE.UTF-8')
'de_DE.UTF-8'
>>> format(12345678.9876, '18.10n')
'      12.345.678,99'
>>> n, t = 15, 42
>>> '{}/{ } ({:.1%})'.format(n, t, float(n)/t)
'15/42 (35.7%)'
```

Notas e Referências

1. No método **str.format**, marcas de substituição podem ser aninhadas na **Especificação do formato** (após o sinal :). Por ex. **{a:{b}}**
2. Para exibir { e } literalmente, use {{ e }}.
3. Na função **format** não é permitido usar marcas de substituição {...}, mas a apenas a sintaxe de **Especificação do formato** descrita nesta página.
4. A opção de **tipo n** depende da configuração de **locale** para funcionar, e o **locale pt_BR** não tem separadores de milhares, por isso usamos **de_DE** no exemplo acima.

PEP 3101 - Advanced String Formatting
<http://bit.ly/pep3101>

Format String Syntax (Documentação do Python 2.7)
<http://bit.ly/fmrsyntax>

Python Essential Reference 4th ed. de David Beazley,
ISBN 978-0672329784 <http://bit.ly/pyeref4>