**python格式化字符串的三种方法（%，format，f-string)**

到目前为止，我所知道的，python格式化字符串有三种方法，第一是早期就有的%，其次是2.5之后的format(),还有就是3.6添加的f字符串调试

**7.1 %格式化字符串**

%格式化字符串是python最早的，也是能兼容所有版本的一种字符串格式化方法，在一些python早期的库中，建议使用%格式化方式，他会把字符串中的格式化符按顺序后面参数替换，格式是

"xxxxxx %s xxxxxx" % (value1, value2)

* 其中 %s就是格式化符，意思是把后面的值格式化为字符类型，类似的格式化符还有%d,%f等，具体参考文章[Python字符串格式化](https://www.cnblogs.com/vitrox/p/4504899.html)
* 后面的value1,value2就是要格式化的值，不论是字符还是数值，都会被格式化为格式化符对应的类型
* 当然可以不用以元组的形式传值，你可以直接写这样："xxxxx %s" % value，不过不建议这样写，一是应为这样只能传递一个参数，二是如果value是元组或列表等类型，这样会触发TypeErrer
* 如果只传一个参数，并且很确定参数类型不会触发异常，可以使用上面的写法，否则，我建议你提供一个单元素元组，就像"xxxx %s " % (value,)

value1 = (7, 8)

value2 = [9, 0]

print("DAY %s 格式化字符串 %s " % (value1,value2))

value3 = 1

s = "xxxix %s" % value3 # 不推荐

print(s)

s1 = "xxxx %s " % value1

print(s1) # TypeError: not all arguments converted during string formatting

**7.2 format()**

%虽然强大，但用起来难免有些麻烦，代码也不是特别美观，因此，在python 2.5 之后，提供了更加优雅的str.format()方法。

def format(self, \*args, \*\*kwargs): # known special case of str.format

"""

S.format(\*args, \*\*kwargs) -> str

Return a formatted version of S, using substitutions from args and kwargs.

The substitutions are identified by braces ('{' and '}').

"""

pass

* format()的常用用法

# 使用名称占位符

s2 = "xxxx {age} xxxx {name}".format(age=18, name="hangman")

print(s2) # xxxx 18 xxxx hangman

# 使用序号占位符，为空默认从左到右01234.。。

s3 = "xxxx {1} xxx{0}".format(value1,value2)

print(s3) # xxxx [9, 0] xxx(7, 8)

# 也可以混合使用

s4 = "xxxx {} XXX {name} xxx {}".format(value2,value1,name="s4")

print(s4) # xxxx [9, 0] XXX s4 xxx (7, 8)

**7.3 f-string**

f-string是2015年python 3.6 根据PEP 498新添加的一种字符串格式化方法，f-string实际上是在运行时计算的表达式，而不是常量值。在Python源代码中，f-string是一个文字字符串，前缀为’f’，其中包含大括号内的表达式。表达式会将大括号中的内容替换为其值。例如

import datetime

name = "zings"

age = 17

date = datetime.date(2019,7,18)

print(f'my name is {name}, this year is {date:%Y},Next year, I\'m {age+1}') # my name is zings, this year is 2019,Next year, I'm 18

**7.3.2 格式规范迷你语言**

“格式规范”用于格式字符串中包含的替换字段中，以定义各个值的显示方式

标准格式说明符的一般形式是：

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" | "%"

**(1) 对齐 align**

| **align（对齐方式）** | **作用** |
| --- | --- |
| < | 左对齐（字符串默认对齐方式） |
| > | 右对齐（数值默认对齐方式） |
| = | 填充时强制在正负号与数字之间进行填充，只支持对数字的填充 |
| ^ | 居中 |

* 除非定义了最小字段宽度，否则字段宽度将始终与填充它的数据大小相同，因此在这种情况下，对齐选项没有意义。
* 如果指定了align值，则可以在其前面加上可以是任何字符的填充字符，如果省略则默认为空格。 无法使用文字大括号（“{”或“}”）作为格式化字符串文字中的填充字符或使用str.format（）方法。 但是，可以插入带有嵌套替换字段的大括号。

print(f'{name:^18}') # | zings |

* 1

**(2) sign**

sign只对数字起作用

| **sign** | **作用** |
| --- | --- |
| + | 强制对数字使用正负号 |
| - | 仅对负数使用前导负号(默认) |
| 空格 | 对正数使用一个’ ‘作前导，负数仍以’-'为前导 |

print(f'{money:+}') # +19999999877

* 1

**(3) #选项**

'＃'选项使“替代形式”用于转换。 对于不同类型，替代形式的定义不同。 此选项仅对integer，float，complex和Decimal类型有效。 对于整数，当使用二进制，八进制或十六进制输出时，此选项将前缀“0b”，“0o”或“0x”添加到输出值。 对于浮点数，复数和十进制，替换形式会导致转换结果始终包含小数点字符，即使后面没有数字也是如此。 通常，只有在跟随数字的情况下，这些转换的结果中才会出现小数点字符。 此外，对于“g”和“G”转换，不会从结果中删除尾随零。

**(4) ,选项**

','被用来对数字整数部分进行千分位分隔

| **描述符** | **作用** |
| --- | --- |
| , | 使用,作为千位分隔符 |
| \_ | 使用\_作为千位分隔符 |

* , 仅适用于浮点数、复数与十进制整数：对于浮点数和复数，, 只分隔小数点前的数位。
* \_ 适用于浮点数、复数与二、八、十、十六进制整数：对于浮点数和复数，\_ 只分隔小数点前的数位；对于二、八、十六进制整数，固定从低位到高位每隔四位插入一个 \_（十进制整数是每隔三位插入一个 \_）。

print(f'{money:,}') # 19,999,999,877

* 1

**(5) width**

width是定义最小字段宽度的十进制整数。 如果未指定，则字段宽度将由内容确定。

当然，format还有很多彪悍的特性，还可以看这位大佬的文章：[Python字符串格式化](https://www.cnblogs.com/vitrox/p/4504899.html)

**(6) .precision**

.precision对于数字对象，用来指定数字的小数位数，如果有小数；对于非数字对象，用来指定最终返回的格式化字符的最大长度，即格式化完成后，以这个precision参数对结果进行截取

**(7) type**

[外链图片转存失败(img-AtydwvFj-1563463465332)(…/…/image/python\_总结\_01.png)]

\*\*注意：\*\*格式规范迷你语言对format一样适用（本来就是format的）

**7.4 总结**

python最先的格式化字符串方法是%，但他的致命缺点是支持的类型有限，只支持int,str,double,其他所有类型只能转换为这几个类型，还有如果传递的是元组，那么必须还要传入一个单值元组，为此，添加了str.format（）以解决％-formatting中的一些问题，特别是，它使用普通函数调用语法（并因此支持多个参数），并且可以通过\_\_format \_\_（）方法在被转换为字符串的对象上进行扩展。但str.format（）又存在代码冗余的问题，例如

v = 6\*8

print("the value is {}".format(v))

* 1
* 2

而使用f-string只需要

print(f'the value is{6\*8}')

* 1

F字符串提供了一种简洁易读的方式，可以在字符串中包含Python表达式的值。包括lambda表达式（要放在括号里）

参考文章

[PEP 498](https://www.python.org/dev/peps/pep-0498/)

[python doc](https://docs.python.org/3/library/string.html#string.Formatter)

[Python字符串格式化](https://www.cnblogs.com/vitrox/p/4504899.html)

[Python格式化字符串f-string概览](https://blog.csdn.net/sunxb10/article/details/81036693)

[GitHub python 面试题](https://github.com/taizilongxu/interview_python#8-%E5%AD%97%E7%AC%A6%E4%B8%B2%E6%A0%BC%E5%BC%8F%E5%8C%96%E5%92%8Cformat)

[Python入门技能树其他Python编程基础](https://edu.csdn.net/skill/python/python-3-114)