PYTHON教程
1. 简介
2. Python历史
3. 安装Python
3.1. Python解释器
4. 第一个Python程序
4.1. 使用文本编辑器
4.2. 输入和输出
5. Python基础
5.1. 数据类型和变量
5.2. 字符串和编码
5.3. 使用list和tuple
5.4. 条件判断
5.5. 模式匹配
5.6. 循环
5.7. 使用dict和set
6. 函数
7. 高级特性
7.1. 切片
7.2. 迭代
7.3. 列表生成式
7.4. 生成器
7.5. 迭代器
8. 函数式编程
9. 模块
10. 面向对象编程
11. 面向对象高级编程
12. 错误、调试和测试
13. IO编程
14. 进程和线程
15. 正则表达式
16. 常用内建模块
17. 常用第三方模块
18. 图形界面
19. 网络编程
20. 电子邮件
21. 访问数据库
22. Web开发
23. 异步IO
24. FAQ
25. 期末总结

字符串和编码

廖雪峰
资深软件开发工程师，业余马拉松选手。

字符编码

我们已经讲过了，字符串也是一种数据类型，但是，字符串比较特殊的是还有一个编码问题。

因为计算机只能处理数字，如果要处理文本，就必须先把文本转换为数字才能处理。最早的计算机在设计时采用8个比特（bit）作为一个字节（byte），所以，一个字节能表示的最大的整数就是255（二进制11111111=十进制255），如果要表示更大的整数，就必须用更多的字节。比如两个字节可以表示的最大整数是 `65535`，4个字节可以表示的最大整数是 `4294967295`。

由于计算机是美国人发明的，因此，最早只有127个字符被编码到计算机里，也就是大小写英文字母、数字和一些符号，这个编码表被称为 `ASCII` 编码，比如大写字母 `A` 的编码是 `65`，小写字母 `z` 的编码是 `122`。

但是要处理中文显然一个字节是不够的，至少需要两个字节，而且还不能和ASCII编码冲突，所以，中国制定了 `GB2312` 编码，用来把中文编进去。

你可以想得到的是，全世界有上百种语言，日本把日文编到 `Shift_JIS` 里，韩国把韩文编到 `Euc-kr` 里，各国有各国的标准，就会不可避免地出现冲突，结果就是，在多语言混合的文本中，显示出来会有乱码。

因此，Unicode字符集应运而生。Unicode把所有语言都统一到一套编码里，这样就不会再有乱码问题了。

Unicode标准也在不断发展，但最常用的是UCS-16编码，用两个字节表示一个字符（如果要用到非常偏僻的字符，就需要4个字节）。现代操作系统和大多数编程语言都直接支持Unicode。

现在，捋一捋ASCII编码和Unicode编码的区别：ASCII编码是1个字节，而Unicode编码通常是2个字节。

字母 `A` 用ASCII编码是十进制的 `65`，二进制的 `01000001`；

字符 `0` 用ASCII编码是十进制的 `48`，二进制的 `00110000`，注意字符 `'0'` 和整数 `0` 是不同的；

汉字 `中` 已经超出了ASCII编码的范围，用Unicode编码是十进制的 `20013`，二进制的 `01001110 00101101`。

你可以猜测，如果把ASCII编码的 `A` 用Unicode编码，只需要在前面补0就可以，因此，`A` 的Unicode编码是 `00000000 01000001`。

新的问题又出现了：如果统一成Unicode编码，乱码问题从此消失了。但是，如果你写的文本基本上全部是英文的话，用Unicode编码比ASCII编码需要多一倍的存储空间，在存储和传输上就十分不划算。

所以，本着节约的精神，又出现了把Unicode编码转化为“可变长编码”的 `UTF-8` 编码。UTF-8编码把一个Unicode字符根据不同的数字大小编码成1-6个字节，常用的英文字母被编码成1个字节，汉字通常是3个字节，只有很生僻的字符才会被编码成4-6个字节。如果你要传输的文本包含大量英文字符，用UTF-8编码就能节省空间：

字符	ASCII	Unicode	UTF-8
A	01000001	00000000 01000001	01000001
中		01001110 00101101	11100100 10111000 10101101

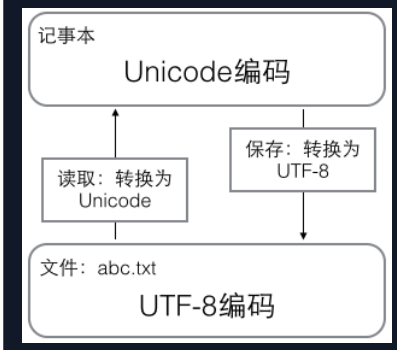
从上面的表格还可以发现，UTF-8编码有一个额外的好处，就是ASCII编码实际上可以被看成是UTF-8编码的一部分，所以，大量只支持ASCII编码的历史遗留软件可以在UTF-8编码下继续工作。

搞清楚了ASCII、Unicode和UTF-8的关系，我们就可以总结一下现在计算机系统通用的字符编码工作方式：

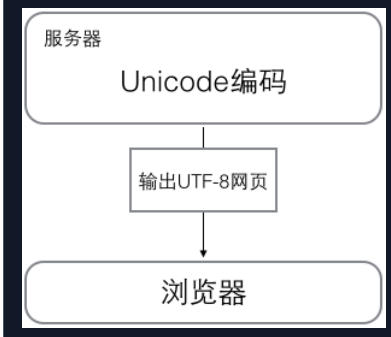
PYTHON教程
1. 简介
2. Python历史
3. 安装Python
3.1. Python解释器
4. 第一个Python程序
4.1. 使用文本编辑器
4.2. 输入和输出
5. Python基础
5.1. 数据类型和变量
5.2. 字符串和编码
5.3. 使用list和tuple
5.4. 条件判断
5.5. 模式匹配
5.6. 循环
5.7. 使用dict和set
6. 函数
7. 高级特性
7.1. 切片
7.2. 迭代
7.3. 列表生成式
7.4. 生成器
7.5. 迭代器
8. 函数式编程
9. 模块
10. 面向对象编程
11. 面向对象高级编程
12. 错误、调试和测试
13. IO编程
14. 进程和线程
15. 正则表达式
16. 常用内建模块
17. 常用第三方模块
18. 图形界面
19. 网络编程
20. 电子邮件
21. 访问数据库
22. Web开发
23. 异步IO
24. FAQ
25. 期末总结

在计算机内存中，统一使用Unicode编码，当需要保存到硬盘或者需要传输的时候，就转换为UTF-8编码。

用记事本编辑的时候，从文件读取的UTF-8字符被转换为Unicode字符到内存里，编辑完成后，保存的时候再把Unicode转换为UTF-8保存到文件：



浏览网页的时候，服务器会把动态生成的Unicode内容转换为UTF-8再传输到浏览器：



所以你看到很多网页的源码上会有类似 `<meta charset="UTF-8" />` 的信息，表示该网页正是用的UTF-8编码。

Python的字符串

搞清楚了令人头疼的字符编码问题后，我们再来研究Python的字符串。

在最新的Python 3版本中，字符串是以Unicode编码的，也就是说，Python的字符串支持多语言，例如：

```
>>> print('包含中文的str')
包含中文的str
```

对于单个字符的编码，Python提供了 `ord()` 函数获取字符的整数表示， `chr()` 函数把编码转换为对应的字符：

```
>>> ord('A')
65
>>> ord('中')
20013
>>> chr(66)
'B'
>>> chr(25991)
'文'
```

如果知道字符的整数编码，还可以用十六进制这么写 `str` ：

```
>>> '\u4e2d\u6587'
'中文'
```

两种写法完全是等价的。

由于Python的字符串类型是 `str` ，在内存中以Unicode表示，一个字符对应若干个字节。如果要在网络上传输，或者保存到磁盘上，就需要把 `str` 变为以字节为单位的 `bytes` 。

Python对 `bytes` 类型的数据用带 `b` 前缀的单引号或双引号表示：

```
x = b'ABC'
```

要注意区分 `'ABC'` 和 `b'ABC'` ，前者是 `str` ，后者虽然内容显示得和前者一样，但 `bytes` 的每个字符都只占用一个字节。

以Unicode表示的 `str` 通过 `encode()` 方法可以编码为指定的 `bytes` ，例如：

PYTHON教程
1. 简介
2. Python历史
3. 安装Python
3.1. Python解释器
4. 第一个Python程序
4.1. 使用文本编辑器
4.2. 输入和输出
5. Python基础
5.1. 数据类型和变量
5.2. 字符串和编码
5.3. 使用list和tuple
5.4. 条件判断
5.5. 模式匹配
5.6. 循环
5.7. 使用dict和set
6. 函数
7. 高级特性
7.1. 切片
7.2. 迭代
7.3. 列表生成式
7.4. 生成器
7.5. 迭代器
8. 函数式编程
9. 模块
10. 面向对象编程
11. 面向对象高级编程
12. 错误、调试和测试
13. IO编程
14. 进程和线程
15. 正则表达式
16. 常用内建模块
17. 常用第三方模块
18. 图形界面
19. 网络编程
20. 电子邮件
21. 访问数据库
22. Web开发
23. 异步IO
24. FAQ
25. 期末总结

```
>>> 'ABC'.encode('ascii')
b'ABC'
>>> '中文'.encode('utf-8')
b'\xe4\xb8\xad\xe6\x96\x87'
>>> '中文'.encode('ascii')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-1: ordinal not in range
```

纯英文的 `str` 可以用 `ASCII` 编码为 `bytes`，内容是一样的，含有中文的 `str` 可以用 `UTF-8` 编码为 `bytes`。含有中文的 `str` 无法用 `ASCII` 编码，因为中文编码的范围超过了 `ASCII` 编码的范围，Python会报错。

在 `bytes` 中，无法显示为ASCII字符的字节，用 `\x##` 显示。

反过来，如果我们从网络或磁盘上读取了字节流，那么读到的数据就是 `bytes`。要把 `bytes` 变为 `str`，就需要用 `decode()` 方法：

```
>>> b'ABC'.decode('ascii')
'ABC'
>>> b'\xe4\xb8\xad\xe6\x96\x87'.decode('utf-8')
'中文'
```

如果 `bytes` 中包含无法解码的字节，`decode()` 方法会报错：

```
>>> b'\xe4\xb8\xad\xff'.decode('utf-8')
Traceback (most recent call last):
  ...
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xff in position 3: invalid start byte
```

如果 `bytes` 中只有一小部分无效的字节，可以传入 `errors='ignore'` 忽略错误的字节：

```
>>> b'\xe4\xb8\xad\xff'.decode('utf-8', errors='ignore')
'中'
```

要计算 `str` 包含多少个字符，可以用 `len()` 函数：

```
>>> len('ABC')
3
>>> len('中文')
2
```

`len()` 函数计算的是 `str` 的字符数，如果换成 `bytes`，`len()` 函数就计算字节数：

```
>>> len(b'ABC')
3
>>> len(b'\xe4\xb8\xad\xe6\x96\x87')
6
>>> len('中文'.encode('utf-8'))
6
```

可见，1个中文字符经过UTF-8编码后通常会占用3个字节，而1个英文字符只占用1个字节。

在操作字符串时，我们经常遇到 `str` 和 `bytes` 的互相转换。为了避免乱码问题，应当始终坚持使用UTF-8编码对 `str` 和 `bytes` 进行转换。

由于Python源代码也是一个文本文件，所以，当你的源代码中包含中文的时候，在保存源代码时，就需要务必指定保存为UTF-8编码。当Python解释器读取源代码时，为了让它按UTF-8编码读取，我们通常在文件开头写上这两行：

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

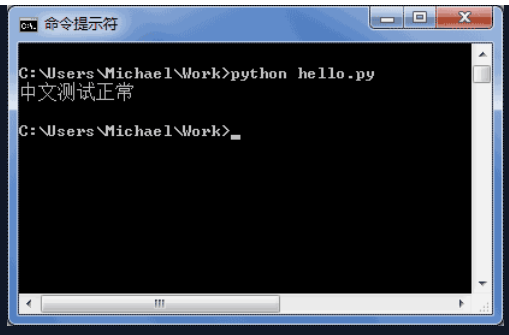
第一行注释是为了告诉Linux/OS X系统，这是一个Python可执行程序，Windows系统会忽略这个注释；

第二行注释是为了告诉Python解释器，按照UTF-8编码读取源代码，否则，你在源代码中写的中文输出可能会有乱码。

申明了UTF-8编码并不意味着你的 `.py` 文件就是UTF-8编码的，必须并且要确保文本编辑器正在使用UTF-8编码。

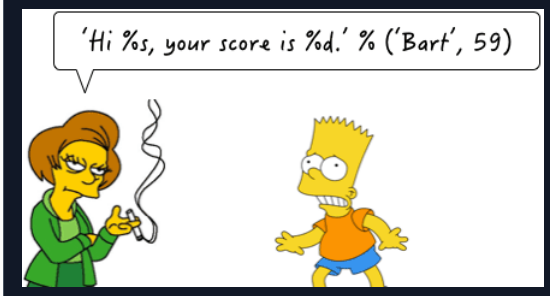
如果 `.py` 文件本身使用UTF-8编码，并且也申明了 `# -*- coding: utf-8 -*-`，打开命令提示符测试就可以正常显示中文：

PYTHON教程
1. 简介
2. Python历史
3. 安装Python
3.1. Python解释器
4. 第一个Python程序
4.1. 使用文本编辑器
4.2. 输入和输出
5. Python基础
5.1. 数据类型和变量
5.2. 字符串和编码
5.3. 使用list和tuple
5.4. 条件判断
5.5. 模式匹配
5.6. 循环
5.7. 使用dict和set
6. 函数
7. 高级特性
7.1. 切片
7.2. 迭代
7.3. 列表生成式
7.4. 生成器
7.5. 迭代器
8. 函数式编程
9. 模块
10. 面向对象编程
11. 面向对象高级编程
12. 错误、调试和测试
13. IO编程
14. 进程和线程
15. 正则表达式
16. 常用内建模块
17. 常用第三方模块
18. 图形界面
19. 网络编程
20. 电子邮件
21. 访问数据库
22. Web开发
23. 异步IO
24. FAQ
25. 期末总结



格式化

最后一个常见的问题是如何输出格式化的字符串。我们经常会输出类似 `'亲爱的xxx你好！你xx月的话费是xx，余额是xx'` 之类的字符串，而xxx的内容都是根据变量变化的，所以，需要一种简便的格式化字符串的方式。



在Python中，采用的格式化方式和C语言是一致的，用 `%` 实现，举例如下：

```
>>> 'Hello, %s' % 'world'
'Hello, world'
>>> 'Hi, %s, you have $%d.' % ('Michael', 1000000)
'Hi, Michael, you have $1000000.'
```

你可能猜到了， `%` 运算符就是用来格式化字符串的。在字符串内部， `%s` 表示用字符串替换， `%d` 表示用整数替换，有几个 `%?` 占位符，后面就跟几个变量或者值，顺序要对应好。如果只有一个 `%?` ，括号可以省略。

常见的占位符有：

占位符	替换内容
%d	整数
%f	浮点数
%s	字符串
%x	十六进制整数

其中，格式化整数和浮点数还可以指定是否补0和整数与小数的位数：

```
print('%2d-%02d' % (3, 1))
print('%.2f' % 3.1415926)
```

如果你不太确定应该用什么， `%s` 永远起作用，它会把任何数据类型转换为字符串：

```
>>> 'Age: %s. Gender: %s' % (25, True)
'Age: 25. Gender: True'
```

有些时候，字符串里面的 `%` 是一个普通字符怎么办？这个时候就需要转义，用 `%%` 来表示一个 `%` ：

```
>>> 'growth rate: %d %%' % 7
'growth rate: 7 %'
```

format()

另一种格式化字符串的方法是使用字符串的 `format()` 方法，它会用传入的参数依次替换字符串内的占位符 `{0}` 、 `{1}` ……，不过这种方式写起来比%要麻烦得多：

PYTHON教程
1. 简介
2. Python历史
3. 安装Python
3.1. Python解释器
4. 第一个Python程序
4.1. 使用文本编辑器
4.2. 输入和输出
5. Python基础
5.1. 数据类型和变量
5.2. 字符串和编码
5.3. 使用list和tuple
5.4. 条件判断
5.5. 模式匹配
5.6. 循环
5.7. 使用dict和set
6. 函数
7. 高级特性
7.1. 切片
7.2. 迭代
7.3. 列表生成式
7.4. 生成器
7.5. 迭代器
8. 函数式编程
9. 模块
10. 面向对象编程
11. 面向对象高级编程
12. 错误、调试和测试
13. IO编程
14. 进程和线程
15. 正则表达式
16. 常用内建模块
17. 常用第三方模块
18. 图形界面
19. 网络编程
20. 电子邮件
21. 访问数据库
22. Web开发
23. 异步IO
24. FAQ
25. 期末总结

```
>>> 'Hello, {0}, 成绩提升了 {1:.1f}%'.format('小明', 17.125)
'Hello, 小明, 成绩提升了 17.1%'
```

f-string

最后一种格式化字符串的方法是使用以 `f` 开头的字符串，称之为 `f-string` ，它和普通字符串不同之处在于，字符串如果包含 `{xxx}` ，就会以对应的变量替换：

```
>>> r = 2.5
>>> s = 3.14 * r ** 2
>>> print(f'The area of a circle with radius {r} is {s:.2f}')
The area of a circle with radius 2.5 is 19.62
```

上述代码中， `{r}` 被变量 `r` 的值替换， `{s:.2f}` 被变量 `s` 的值替换，并且 `:` 后面的 `.2f` 指定了格式化参数（即保留两位小数），因此， `{s:.2f}` 的替换结果是 `19.62` 。

练习

小明的成绩从去年的72分提升到了今年的85分，请计算小明成绩提升的百分点，并用字符串格式化显示出 `'xx.x%'` ，只保留小数点后1位：

```
s1 = 72
s2 = 85
r = ???
print('???')
```

参考源码

the_string.py

小结

Python 3的字符串使用Unicode，直接支持多语言。

当 `str` 和 `bytes` 互相转换时，需要指定编码。最常用的编码是 `UTF-8` 。Python当然也支持其他编码方式，比如把Unicode编码成 `GB2312` ：

```
>>> '中文'.encode('gb2312')
b'\xd6\xd0\xce\xc4'
```

但这种方式纯属自找麻烦，如果没有特殊业务要求，请牢记仅使用 `UTF-8` 编码。

格式化字符串的时候，可以用Python的交互式环境测试，方便快捷。

« 数据类型和变量

使用list和tuple »

Comments

Comments loaded. To post a comment, please Sign In

. @ 2026/1/1 15:28:15

s1 = 72

PYTHON教程
1. 简介
2. Python历史
3. 安装Python
3.1. Python解释器
4. 第一个Python程序
4.1. 使用文本编辑器
4.2. 输入和输出
5. Python基础
5.1. 数据类型和变量
5.2. 字符串和编码
5.3. 使用list和tuple
5.4. 条件判断
5.5. 模式匹配
5.6. 循环
5.7. 使用dict和set
6. 函数
7. 高级特性
7.1. 切片
7.2. 迭代
7.3. 列表生成式
7.4. 生成器
7.5. 迭代器
8. 函数式编程
9. 模块
10. 面向对象编程
11. 面向对象高级编程
12. 错误、调试和测试
13. IO编程
14. 进程和线程
15. 正则表达式
16. 常用内建模块
17. 常用第三方模块
18. 图形界面
19. 网络编程
20. 电子邮件
21. 访问数据库
22. Web开发
23. 异步IO
24. FAQ
25. 期末总结

s2 = 85

r = (s2 - s1) / s1 * 100

print(f'The percentage of growth in grade is {r:.1f}%')

The percentage of growth in grade is 18.1%

仰望星空 @ 2025/12/31 01:30:20

s1 = 72 s2 = 85 r = s2/s1 print(f"{r:.1f}%")

张肥肥 @ 2025/12/28 05:46:27

s1=72 s2=85 r=100*(s2-s1)/s1 print(f'小明的成绩提升了{r:.2f}%')

> 
>
> 张肥肥 @ 2025/12/28 05:57:42
>
> > *s1=72 s2=85 r=100*(s2-s1)/s1 print('小明成绩提升了%.2f%%' % r) 小明成绩提升了18.06% print('小明成绩提升了{0:.2f}%'.format(r)) 小明成绩提升了18.06% print(f'小明的成绩提升了{r:.2f}%') 小明的成绩提升了18.06%*
>
> 
>
> 张肥肥 @ 2025/12/28 06:02:14
>
> print('{0}的成绩提升了{1:.2f}%'.format('小明',r)) 小明的成绩提升了18.06%

breeze @ 2025/12/26 08:30:19

打卡第三天，5.2节结束。 抄了答案，看着答案才能理解，再接再厉，加油!

时光过得好快 @ 2025/12/25 18:47:25

s1 = 72

s2 = 85

e = ((s2-s1)/s2*100)

a = '小明的成绩提升了: ' '%.1f %%' % e

print(a)

b = '{0} {1:.1f} %'.format('小明的成绩提升了:' ,e)

print(b)

c = f'小明的成绩提升了: {e:.1f} %'

print(c)

▵→奥霍斯德尔萨拉多 @ 2025/12/14 06:51:59

Excel里用的TEXT函数和这个有点像

```
>>>>>> print('%2d-%02d'%(3,1))
 3-01
>>> print('%02d-%02d'% (3,1))
03-01
>>> print('%03d-%02d'%(3,1))
003-01
>>> print('%3d-%02d'%(3,1))
  3-01
>>> print('%x6d-%04d'%(3,1))
36d-0001
>>> print('%06d-%08d'%(3,1))
000003-00000001
```

PYTHON教程
1. 简介
2. Python历史
3. 安装Python
3.1. Python解释器
4. 第一个Python程序
4.1. 使用文本编辑器
4.2. 输入和输出
5. Python基础
5.1. 数据类型和变量
5.2. 字符串和编码
5.3. 使用list和tuple
5.4. 条件判断
5.5. 模式匹配
5.6. 循环
5.7. 使用dict和set
6. 函数
7. 高级特性
7.1. 切片
7.2. 迭代
7.3. 列表生成式
7.4. 生成器
7.5. 迭代器
8. 函数式编程
9. 模块
10. 面向对象编程
11. 面向对象高级编程
12. 错误、调试和测试
13. IO编程
14. 进程和线程
15. 正则表达式
16. 常用内建模块
17. 常用第三方模块
18. 图形界面
19. 网络编程
20. 电子邮件
21. 访问数据库
22. Web开发
23. 异步IO
24. FAQ
25. 期末总结

```
>>> print('%6d-%8d'%(3,1))
     3-       1
```

△→奥霍斯德尔萨拉多 @ 2025/12/15 01:42:46

测试可行

```
>>> s1=72
>>> s2=85
>>> r=(s2-s1)/s1*100
>>> print('小明成绩提升了%.1f%%' % r)
小明成绩提升了18.1%
>>> print('小明成绩提升了{0:.1f}%'.format(r))
小明成绩提升了18.1%
>>> print(f'小明成绩提升了{r:.1f}%')
小明成绩提升了18.1%
```

白色的夜晚 @ 2025/12/21 03:26:38

你好，整数不是'%d'吗?为什么写'%2d'呢?

Mooooses @ 2025/12/22 03:40:58

2是表示整数前面空2个空格吧，如果从0开始算应该是1个。我也不太清楚。。。

芭乐香蕉 @ 2025/12/22 20:51:55

@白色的夜晚，%2d表示整数占来两个宽度，不够用空格补充，%02d就是不够就用0补充

☆ @ 2025/12/24 00:27:29

- - coding: utf-8 --

s1 = 72 s2 = 85 r = (s2 - s1) / s1 * 100 print(f'小明去年的成绩是：{s1}，今年的成绩是：{s2},提升百分比：{r:.1f}%')

print("小名去年的成绩是：{0}，今年的成绩是：{1}，提升百分比：{2:.1f}%".format(s1,s2,r))

有何不可 @ 2025/12/15 02:54:06

r = 72 s = 85 print(f'小明的成绩从去年的{r}分提升到了今年的{s}分,提升了{(s-r)/r*100:.1f}%')

me、Tranquility @ 2025/12/14 00:13:18

> *s1=72 s2=85 r=(s2-s1)/s1*100 print('小明成绩提高了%.2f%%'%r) 小明成绩提高了18.06% print('小明成绩提高了{0:.2f}%'.format(r)) 小明成绩提高了18.06% print(f'小明成绩提高了{r:.2f}%') 小明成绩提高了18.06%*

砀峦 @ 2025/12/10 00:56:09

s1 = 72 s2 = 85 r = (s2/s1-1)*100 print(f'小明成绩提升的百分点{r:.1f}%')

北方烟火 @ 2025/12/6 04:39:59

一下是我的3种写法，亲测可行

```
s1 = 72
s2 = 85
r = (s2 - s1)/s1*100
print('小明的成绩提升了:%.1f%%' % r)
print(f'小明的成绩提升了:{r:.1f}%')
print('小明的成绩提升了:{0:.1f}%'.format(r))
```

PYTHON教程
1. 简介
2. Python历史
3. 安装Python
3.1. Python解释器
4. 第一个Python程序
4.1. 使用文本编辑器
4.2. 输入和输出
5. Python基础
5.1. 数据类型和变量
5.2. 字符串和编码
5.3. 使用list和tuple
5.4. 条件判断
5.5. 模式匹配
5.6. 循环
5.7. 使用dict和set
6. 函数
7. 高级特性
7.1. 切片
7.2. 迭代
7.3. 列表生成式
7.4. 生成器
7.5. 迭代器
8. 函数式编程
9. 模块
10. 面向对象编程
11. 面向对象高级编程
12. 错误、调试和测试
13. IO编程
14. 进程和线程
15. 正则表达式
16. 常用内建模块
17. 常用第三方模块
18. 图形界面
19. 网络编程
20. 电子邮件
21. 访问数据库
22. Web开发
23. 异步IO
24. FAQ
25. 期末总结

北方烟火 @ 2025/12/6 04:30:17

```
s1 = 72
s2 = 85
r = (s2 - s1)/s1*100
print('小明的成绩提升了:%.1f%%' % r)
```

某不科学动物 @ 2025/12/5 09:09:17

```
s1=72
s2=85
r=((85-72)/72)*100
print ('小明今年提高了:','%.1f%%' % r)
```

吃货美少女 @ 2025/12/4 03:18:09

s1 = 72 s2 = 85 r = ((s2 - s1) / s1) * 100

```
print('成绩提升了 %.1f%%' % r)
print("成绩提升比例为{:.1f}%".format(r))
print(f'小明的成绩提升比例为{r:.1f}%')
```

土豆泥 @ 2025/12/2 21:57:16

s1 = 72 s2 = 85 r = 100*(s2-s1)/s1 print(f'小明的成绩从去年的{s1}，提升到了今年的{s2},小明的成绩相对去年提升了{r:.1f}%')

BMW @ 2025/12/1 04:54:26

s1 = 72 s2 = 85 r = (s2 - s1)/ s2 * 100 #print(f'{r:.2f}%') print('%.2f%%' %r)

幺鸡 @ 2025/11/30 22:45:39

`print({name} 的成绩提升了 {score:}%")`

没有引号 ⇒ Python 不知道这是字符串

Python 看到 `{name}` 以后会认为那是一个“集合”（set），不是字符串

没有加 f ⇒ Python 不会替换变量

`{name}` 只有在 **f-string** 下才会被替换成变量内容。没有 `f`，它就只是非法语法。

`{score:}` **不是合法格式**

必须写成 `.1f`、`.2f`、`d` 等格式

`:` **分隔符，用来引入格式化指令** `.1f` 就是 **格式化指令**

f-string 的语法规定： `f 必须放在引号前面` **，而不能放在引号里面。**

Read More ▼

啊，你以为我傻 @ 2025/11/27 04:46:46

> *s1 = 72 s2 = 85 r = (s2 - s1)/s1*100 print(f"小明成绩提升了{r:.1f}%.") 小明成绩提升了18.1%.*

Hypersomnia @ 2025/11/18 05:16:46

```
s1 = 72
s2 = 85
r = (s2 - s1)/ s1 *100
print(f'小明成绩提升了{r:.1f}%')
```

PYTHON教程
1. 简介
2. Python历史
3. 安装Python
3.1. Python解释器
4. 第一个Python程序
4.1. 使用文本编辑器
4.2. 输入和输出
5. Python基础
5.1. 数据类型和变量
5.2. **字符串和编码**
5.3. 使用list和tuple
5.4. 条件判断
5.5. 模式匹配
5.6. 循环
5.7. 使用dict和set
6. 函数
7. 高级特性
7.1. 切片
7.2. 迭代
7.3. 列表生成式
7.4. 生成器
7.5. 迭代器
8. 函数式编程
9. 模块
10. 面向对象编程
11. 面向对象高级编程
12. 错误、调试和测试
13. IO编程
14. 进程和线程
15. 正则表达式
16. 常用内建模块
17. 常用第三方模块
18. 图形界面
19. 网络编程
20. 电子邮件
21. 访问数据库
22. Web开发
23. 异步IO
24. FAQ
25. 期末总结

卿绛 @ 2025/11/17 22:18:24

```
a=72

b=85

// %老式占位符写法

print('小明成绩提升了%.1f%%' %((b-a)/a*100))

// format字符串格式化方法

print('小明成绩提升了{:.1f}%'.format((b-a)/a*100))

// f-string式写法

print(f'小明成绩提升了{(b-a)/a*100:.1f}%')
```