# python中print用法

#### 一、print()函数概述

print() 方法用于打印输出，是[python](https://so.csdn.net/so/search?q=python&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qdPython/article/details/_blank)中最常见的一个函数。

该函数的语法如下：

print(\*objects, sep=' ', end='\n', file=sys.stdout)

参数的具体含义如下：

objects --表示输出的对象。输出多个对象时，需要用 , （逗号）分隔。

sep – 用来间隔多个对象。

end – 用来设定以什么结尾。默认值是换行符 \n，我们可以换成其他字符。

file – 要写入的文件对象。

print(1) #数值类型可以直接输出

'''

运行结果如下

1

'''

print("Hello World") #字符串类型可以直接输出

'''

运行结果如下：

Hello World

'''

a=1

b="Hello World"

print(a, b) #可以一次输出多个对象，对象之间用逗号分隔

'''

运行结果如下：

1 Hello World

'''

#如果直接输出字符串，而不是用对象表示的话，可以不使用逗号

print("Duan""Yixuan")

print("Duan","Yixuan")

'''

运行结果如下：

DuanYixuan

Duan Yixuan

可知，不添加逗号分隔符，字符串之间没有间隔

'''

print("www", "snh48", "com", sep=".") # 设置间隔符'''

运行结果如下：

www.snh48.com

'''

#### 二、变量的输出

无论什么类型的数据，包括但不局限于：数值型，布尔型，列表变量，字典变量…都可以直接输出。

例如：

num = 19

print(num) #19 输出数值型变量

str = 'Duan Yixuan'

print(str) #Duan Yixuan 输出字符串变量

list = [1,2,'a']

print(list) #[1, 2, 'a'] 输出列表变量

tuple = (1,2,'a')

print(tuple) #(1, 2, 'a') 输出元组变量

dict = {'a':1, 'b':2}

print(dict) # {'a': 1, 'b': 2} 输出字典变量

#### 三、数据的格式化输出

在C语言中，我们可以使用printf("%-.4f",a)之类的形式，实现数据的的格式化输出。

在python中，我们同样可以实现数据的格式化输出。我们可以先看一个简单的例子：

s='Duan Yixuan'

x=len(s)

print('The length of %s is %d' %(s,x))

'''

'The length of %s is %d' 这部分叫做：格式控制符

(s,x) 这部分叫做：转换说明符

% 字符，表示标记转换说明符的开始

输出如下：

The length of Duan Yixuan is 11

'''

和C语言的区别在于，Python中格式控制符和转换说明符用%分隔，C语言中用逗号。

接下来我们仔细探讨一下格式化输出

示例一

#3种不同的格式化方法

print(‘我是{}，我今年{}岁。’.format(name,age))

print(f‘我是{name}，我今年{age}岁。’)

print(‘我是%s，我今年%s岁。’ %(name,age))

示例二

# 对浮点数，保留小数点后几位

print('{:0.3f}'.format(50.5 / 220.5))

# print 格式化字符串

num = int(input('请输入一个十进制的整数：')) # 将str 转为int类型

print(num, '的二进制数为：', bin(num)) # 第一种写法使用了个数可变的位置参数

print(str(num) + '的二进制数为：' + str(bin(num))) # 第二种写法，使用“+”作为连接符 (+的左右均为str类型)

print('%s的二进制数为：%s' % (num, bin(num))) # 第三种写法 格式化字符串

print('\033[0;35m{0}的二进制数为：{1}\033[m'.format(num, bin(num))) # 第三种写法 格式化字符串

print(f'{num}的二进制数为：{bin(num)}') # 第三种写法 格式化字符串

print('-'.center(50, '-'))

print(f'{num}的八进制数为{oct(num)}')

print(f'{num}的十六进制数为{hex(num)}')

#运行结果

0.229

请输入一个十进制的整数：96

96 的二进制数为： 0b1100000

96的二进制数为：0b1100000

96的二进制数为：0b1100000

96的二进制数为：0b1100000

96的二进制数为：0b1100000

--------------------------------------------------

96的八进制数为0o140

96的十六进制数为0x60

**3.1 %字符**

(1)、%字符：标记转换说明符的开始。

%字符的用法可参考上例，不再赘述。

**3.2 最小字段宽度和精度**

最小字段宽度：转换后的字符串至少应该具有该值指定的宽度。如果是\*（星号），则宽度会从值元组中读出。

点(.)后跟精度值：如果需要输出实数，精度值表示出现在小数点后的位数。如果需要输出字符串，那么该数字就表示最大字段宽度。如果是\*，那么精度将从元组中读出。

可参考C语言的实现方式。

注：字段宽度中，小数点也占一位。

PI = 3.141592653

print('%10.3f'%PI) #字段宽10，精度3# 3.142

#精度为3，所以只显示142，指定宽度为10，所以在左边需要补充5个空格，以达到10位的宽度

PI=3.1415926

print("PI=%.\*f" %(3,PI))

#用\*从后面的元组中读取字段宽度或精度,可以读取出来精度是3位

#PI=3.142

#没有指定宽度，所以不需要缩进

print("PI=%\*.3f"%(10,PI)) #精度为3，总长为10.

# PI= 3.142

#\* 所处的位置不同，读取的内容也不同

**3.3 转换标志**

转换标志：

* -表示左对齐；
* +表示在数值前要加上正负号；
* " "(空白字符)表示正数之前保留空格()；
* 0表示转换值若位数不够则用0填充。

具体的我们可以看一下例子：

PI=3.1415926

print('%-10.3f' %PI) #左对齐，还是10个字符，但空格显示在右边。

#3.142

PI=3.1415926

print('%+f' % PI) #显示正负号

#+3.141593# 类型f的默认精度为6位小数。

PI=3.1415926

print('%010.3f'%PI)

#字段宽度为10，精度为3，不足处用0填充空白

#000003.142 0表示转换值若位数不够则用0填充

**3.4 格式字符归纳**

| 格式字符 | 说明 | 格式字符 | 说明 |
| --- | --- | --- | --- |
| %s | 字符串采用str()的显示 | %x | 十六进制整数 |
| %r | 字符串(repr())的显示 | %e | 指数（基底写e） |
| %c | 单个字符 | %E | 指数（基底写E） |
| %b | 二进制整数 | %f,%F | 浮点数 |
| %d | 十进制整数 | %g | 指数(e)或浮点数(根据显示长度) |
| %i | 十进制整数 | %G | 指数(E)或浮点数(根据显示长度) |
| %o | 八进制整数 | %% | 字符% |

#### 四、换行与防止换行

在python中，输出函数总是默认换行，比如说：

for x in range(0,5):

print(x)

'''

0

1

2

3

4

'''

而显然，这种输出太占“空间”，我们可以进行如下改造：

参考文本第一部分对end参数的描述：end – 用来设定以什么结尾。默认值是换行符 \n，我们可以换成其他字符。

for x in range(0, 5):

print(x, end=' ')

#0 1 2 3 4

for x in range(0, 5):

print(x, end=',')

#0,1,2,3,4,

但如果，我们同时运行上面两段代码，结果会如下所示，可知：我们需要在两次输出间，实现换行。

for x in range(0, 5):

print(x, end=' ')

for x in range(0, 5):

print(x, end=',')

#0 1 2 3 4

#0,1,2,3,4,

我们比较以下几种方式

方式一：

for x in range(0, 5):

print(x, end=' ')

print('\n')

for x in range(0, 5):

print(x, end=',')

'''

0 1 2 3 4

0,1,2,3,4,

'''

之所以出现上面这种情况，是因为print()本身就是默认换行的，再加上换行符，相当于换行两次。

方式二：

for x in range(0, 5):

print(x, end=' ')

print() #本身自带换行，完美输出

for x in range(0, 5):

print(x, end=',')

'''

0 1 2 3 4

0,1,2,3,4,

'''