■书籍推荐

<https://www.cnblogs.com/ipaomi/p/7715674.html>

■环境搭建

Anaconda下载地址

<https://www.anaconda.com/download/#download>

安装时注意勾选添加环境变量

安装完命令行打python

退出是exit()

Anaconda提供了显示高亮颜色的命令行，输入ipython即可

■编辑器和IDE

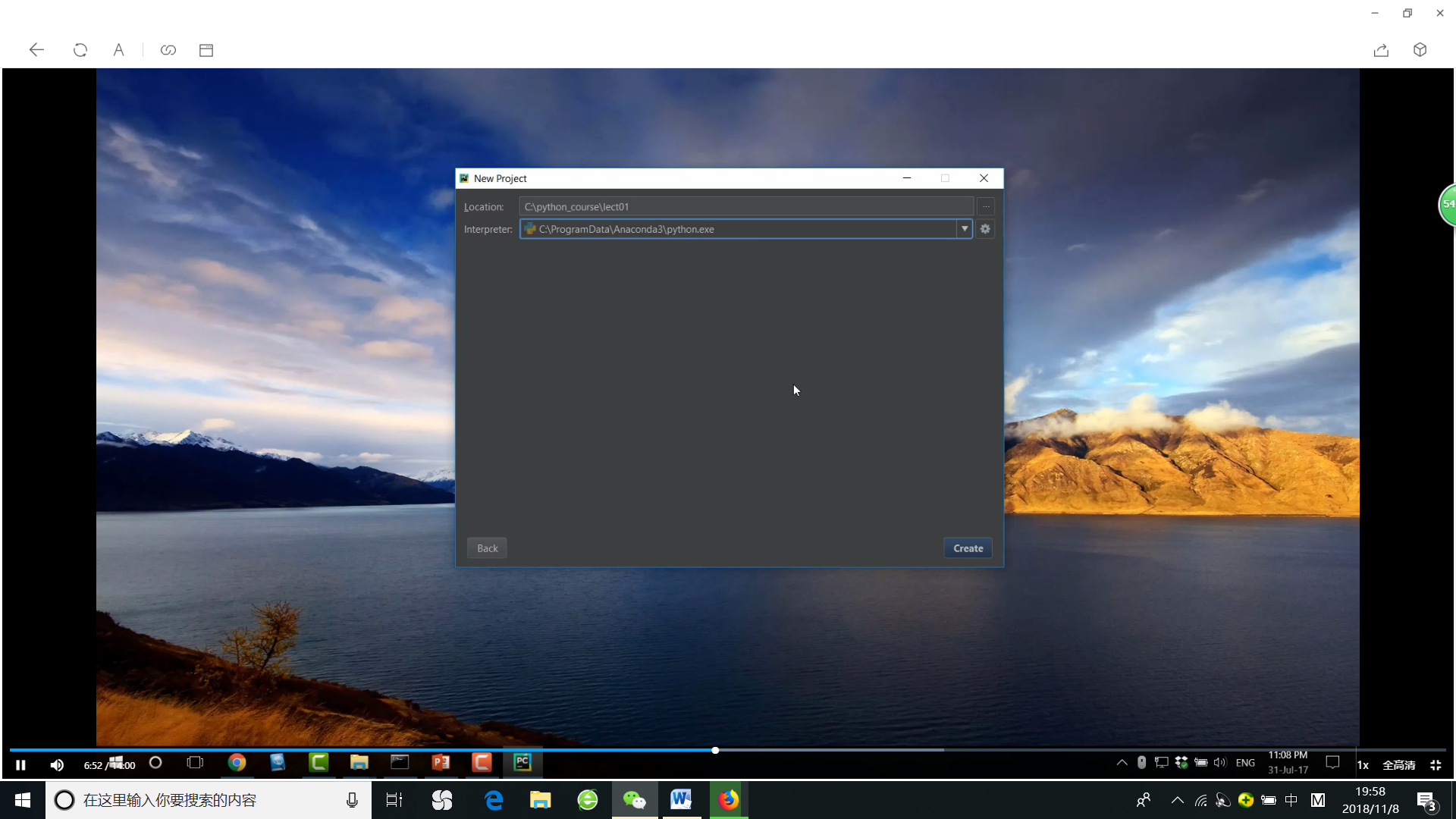
使用anaconda自带的编辑器idle

在命令行输入idle即可

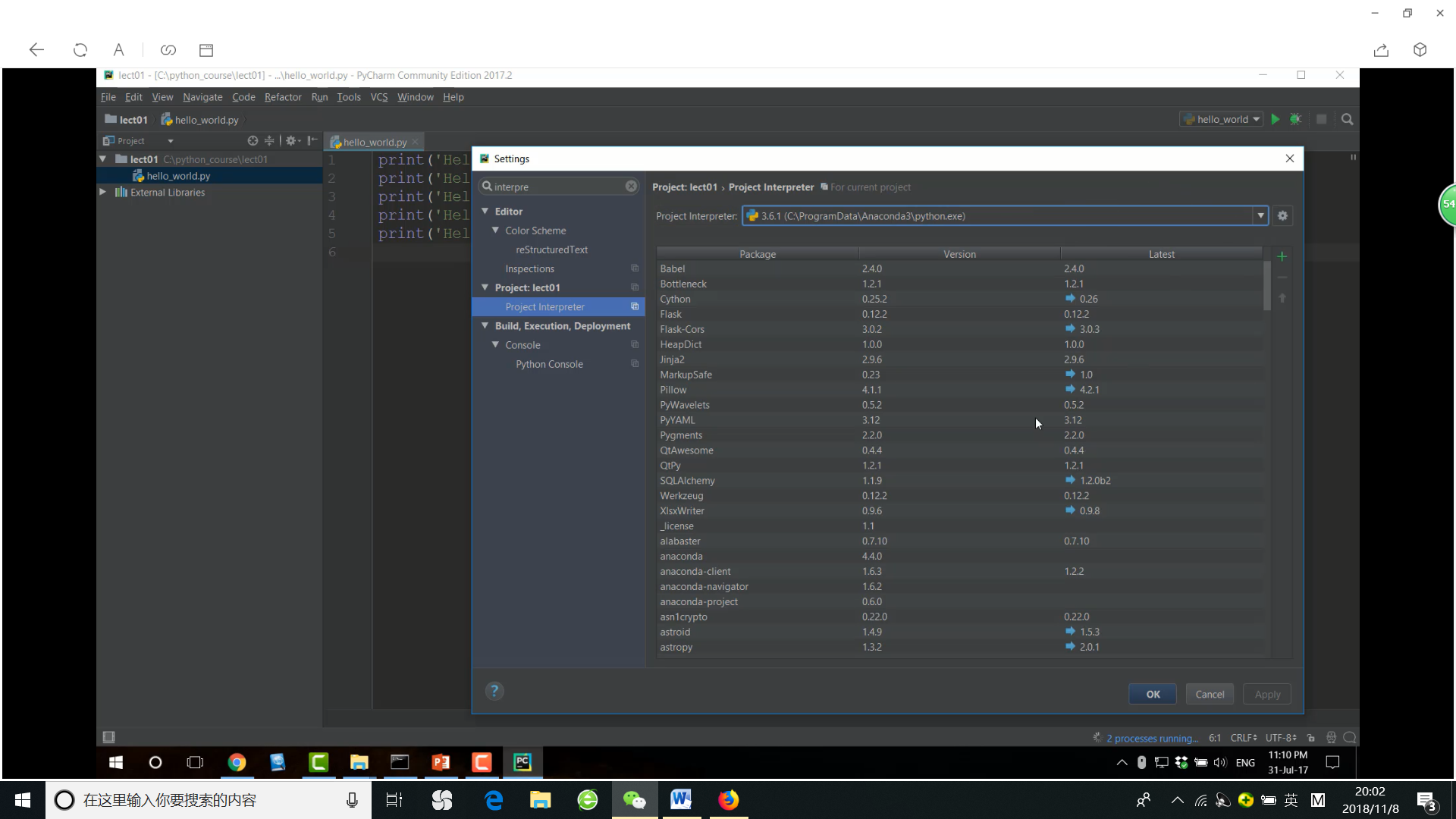
大型程序可以使用pyCharm

<http://www.jetbrains.com/pycharm/>

安装pyCharm后需要配置anaconda解释器



之后在setting里也可以配置



■占位符pass

Pass表示先占位，之后再完善其内容。这样系统不会报错

例如

a = 1  
if a == 1:  
 pass  
else:  
 pass

■输入输出

输入

input()

输出

print()

■注释

快捷键 ctrl + /

单行在开头加 #

多行可以用”””

"""  
这里是多行注释，一般用在文档的开头  
表明一些版本信息等，如：  
测试程序v1.1  
 v1.0更新内容为...  
"""  
  
# 这里是单行注释

■变量和常量

●变量命名推荐

变量名必须是大小写英文、数字和\_的组合，且不能用数字开头！！！

Python对大小写敏感，所以可以使用驼峰

有意义的名字，数据类型，变量标记。中间用下划线隔开

moneyRmb\_str\_val = '100'  
moneyRmb\_num\_val = 100

python是动态语言，也就是弱类型语言，类似javascript

java是静态语言，也就是强类型语言

●常量

通常用全大写字符代表常量。Python没有任何机制保证常量的不变性，因此用全大写命名常量只是提醒程序员这是常量，不要改变它

■缩进

Python缩进只能是4个空格，或者tab 退缩进是shift + tab

必须严格缩进，因为python的缩进用来代表块

■数据类型

●字符串的转义

# 使用\来转义引号  
a = 'I\'m \"ok\".'  
print(a)  
# 输出>>> I'm "ok".

# \n是换行  
print('I\'m learning\nPython.')  
# 输出以下：  
# I'm learning  
# Python.

# \也可以转义它自己，也就是\\  
print('\\\n\\')  
# 输出如下：  
# \  
# \

# \t表示制表符  
print('\\\t\\')  
# 输出：\ \

# r''内的字符串不被转义  
print(r'\\\t\\')  
# 输出\\\t\\

●除法

# 用/是精确除法  
print(10 / 3)  
# 输出：3.3333333333333335  
  
# 用//是取整的除法，也叫地板除  
print(10 // 3)  
# 输出： 3

■字符编码

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；

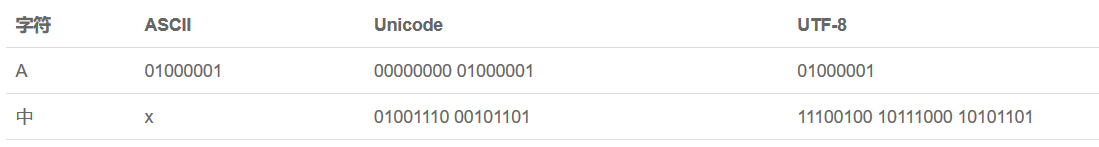
字符0用ASCII编码是十进制的48，二进制的00110000，注意字符'0'和整数0是不同的；

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：

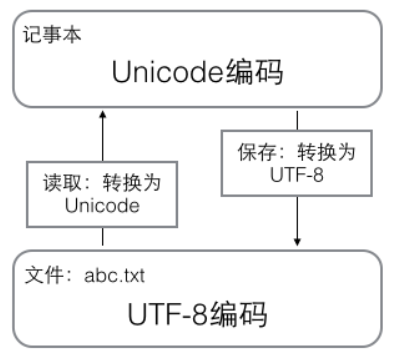


从上面的表格还可以发现，UTF-8编码有一个额外的好处，就是ASCII编码实际上可以被看成是UTF-8编码的一部分，所以，大量只支持ASCII编码的历史遗留软件可以在UTF-8编码下继续工作。

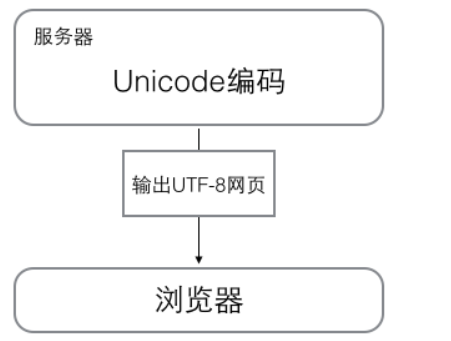
搞清楚了ASCII、Unicode和UTF-8的关系，我们就可以总结一下现在计算机系统通用的字符编码工作方式：

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。

总结一下：

ASCII编码：纯英文，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

Unicode编码：多语言，用Unicode编码比ASCII编码需要多一倍的存储空间

UTF-8编码：中英文。

所以一般的做法是：

计算机的内存中统一使用Unicode编码

当需要保存到硬盘或数据传输时，使用UTF-8编码

■字符串的一些方法

单个字符转编码

# ord()用来将单个字符转化成编码,注意只能转化单个字符  
a = ord('中')  
print(a)  
# 输出： 20013  
  
# chr()用来将单个字符编码转化成字符，注意只能转化单个字符的编码  
b = chr(20013)  
print(b)  
# 输出： 中

Str转bytes（字符串转字节）

# 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。  
# 纯英文的str可以用ASCII编码为bytes。  
# 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。  
  
# 直接用b加引号，表示把引号内的字符串转化成ASCII的bytes  
a = b'ABc'  
print(a)  
# 输出： b'ABc'  
  
# str的encode()方法可以指定转化的编码类型  
print('ABc'.encode('ascii'))  
# 输出： b'ABc'  
print('中文'.encode('utf-8'))  
# 输出： b'\xe4\xb8\xad\xe6\x96\x87'

Betys转str（字节转字符串）

# 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：  
print(b'ABc'.decode('ascii'))  
# 输出：ABc  
print(b'\xe4\xb8\xad\xe6\x96\x87'.decode('utf-8'))  
# 输出： 中文  
  
# 如果bytes中包含无法解码的字节，decode()方法会报错：  
# 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：  
print(b'\xe4\xb8\xad\xff'.decode('utf-8', errors='ignore'))  
# 输出： 中

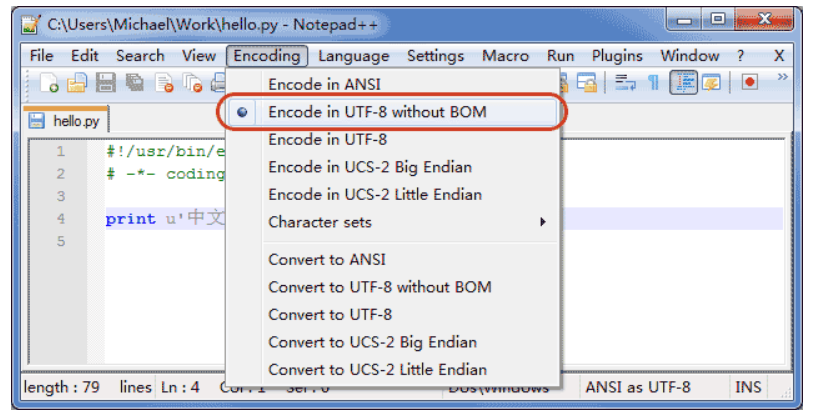
Len()方法

# len()方法用来计算字符串或字节的长度  
print(len('abc'))  
print(len(b'ABC'))  
print(len('中文'.encode('utf-8')))

重要的开头

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

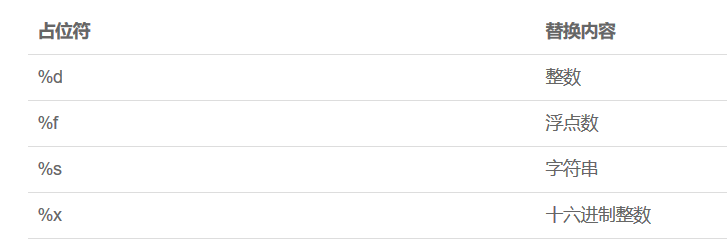
申明了UTF-8编码并不意味着你的.py文件就是UTF-8编码的，必须并且要确保文本编辑器正在使用UTF-8 without BOM编码：



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：

■字符串的格式化

# 字符串中用%s占位，字符串后加% 和括号，括号里依次列出占位的变量的值  
print('hello %s %s' % ('my', 'world'))  
# 输出： hello my world  
  
# 如果只有一个占位的变量，则后面的括号可以省略  
print('hello %s' % 'world')  
# 输出：hello world



# 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))  
# 输出：3-01  
print('%.2f' % 3.1415926)  
# 输出：3.14

# 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：  
print('Age: %s. Gender: %s' % (25, True))  
# 输出： 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%：  
print('growth rate: %d %%' % 7)  
# 输出：growth rate: 7 %

Format()方法

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