python 中的编码

1问题1：问题在哪里?

我们在shell中键入python以打开python命令行，并键入如下两句话：

```
s = "中国zg"
e  = s.encode("utf-8")
```

现在的问题是：**这段代码能运行吗?**

答案是**不能**，会报如下的错：
UnicodeDecodeError: 'ascii' codec can't decode byte **0xe4** in position 0: ordinal not in range(128)

请留意一下错误中说明的0xe4，它是我们分析错误的突破口。

相信很多人都遇到过这个错误。那么新的问题来了。

2问题2：Why?

要搞清楚原因，我们不妨认真分析下这两句话的执行流程：

首先，我们通过键盘在python命令行解释器中键入了 **中国zg** 并且给它加上了英文的双引号，然后又赋值给了变量s，看起来很稀松平常是不是？其实里面大有玄机。

当我们通过键盘在程序中输入字符时，我们是通过操作系统完成这个功能的。我们在屏幕上看到的 **中国zg** 实际上是操作系统给我们人类的一个反馈，告诉你：“嗨，哥们，你在程序中输入了字符 **中国zg** ”

那操作系统给程序的反馈是什么呢？答案就是01串，这个01串是什么样子，又是怎么生成的呢？
答案就是操作系统使用自己的默认编码方式，将**中国zg**进行了编码，并把编码后的01串给了程序。

我们用的centos系统默认的编码是utf-8，所以，只要知道**中国zg**每个字符的utf-8的编码就可以知道01串是什么了。

查询后，可以获得它们的编码是(以16进制和2进制表示)：

中	国	z	g
: ----------	:----------:	-----:
E4B8AD	E59BBD	7A	67

|11100101 10011011 10111101|11100101 10011011 10111101|01111010|01100111|

现在我们知道操作系统传给程序的01串长什么样子了。然后，程序会怎么处理它呢?

程序看到这个01串被双引号包围着，自然知道这个01串是一个字符串。然后这个字符串被赋值给了s。

到此，就是第一句的执行逻辑。

现在继续进行第二句的执行。

e = s.encode("utf-8")的意思是将字符串s用utf-8进行编码，并将编码后的字符串赋值给e。问题来了，程序现在知道s中的01串，还知道这个01串表示的是字符串，但这个字符串的编码是什么呢?

我们必须知道01串的现有编码才能解析出里面的字符，也才能用新的编码方式，如utf-8来重新编码它。操作系统只给程序传来了01串，并没有告诉程序这个01串用的字符编码是什么。

此时，python程序就会用它自己默认的编码当作s的编码，进而来识别s中的内容。这个默认的编码是ASCII，所以，它会用ASCII来解释这个01串，识别出字符串的内容，再将这个字符串转为utf-8编码。

好了，程序碰到的第一个字节就是E4（11100101 ），傻眼！ ASCII编码中没有这玩意儿，因为ASCII编码中字节第一位都是0。

怎么办?
报错呗，于是我们就看到了上面的错误。
错误中的0xe4就是字符 “中” 的utf8编码的第一个字节。

3问题3：How?

知道问题出在哪里了，怎么解决这个问题呢?

显然，我们只要告诉程序，这个s中的01串的编码是utf-8，程序就应该能正确工作。

但这样的解决方法有一个问题，就是不够通用。

假如我有个程序，它要读取很多文本文件，每个文本文件的编码都不一样，岂不是针对每个读进来的文件都维护一个编码信息？很繁琐。

进一步，如果这些文本文件的内容还要做相互的比较连接之类的操作，编码都不一致，岂不是更麻烦?

python是怎么聪明地解决这个问题的呢?

很简单，就是decode!

decode的意思是说，你有一个字符串，并且你知道它的编码，只要你用该编码decode这个字符串，那么，python就会识别出里面的字符内容，同时，建一个int数组，将每个字符的unicode序号存进去。

所有的字符串都这样做，就可以确保在程序运行过程中，各种来源获得的字符串都有一样的表示。它们就可以方便地进行各种操作了。

上面说的 int数组会被python封装成一个对象，即unicode对象。

4问题4：如何搞定?

下面，我们在python命令行中输入如下两行代码:

```
e = s.decode("utf-8")
isinstance(e,unicode)
```

程序的输出是True，这说明，decode后返回的e确实是一个unicode对象。
unicode在这里是一个类，是python里面的类。

e 被称作unicode字符串，意思是说，它存的是字符的unicode序号，并没有使用任何编码。

然后，我们就可以将e编码成任意一种编码，比如下面的操作都是可以的

```
e.encode("utf-8")
e.encode("gbk")
```

只要你选择的编码能够对e中的字符进行编码即可，如果不能编码，就会报错。
比如，如果你尝试这样:

```
e.encode("ascii")
```

由于ASCII并不能编码 中国 这两个字符，所以会爆出 encode error。

至此，我们已经看到了两种错误，decode error 和encode error，并解决了它们。

5问题5：如何评价python的这种字符编码处理方法?

首先，这样的处理方法非常的简单。任何文本，只要它进入程序时进行一次decode，就会变成unicode对象，里面用int存着每个字符的unicode序号。只要在这个文本要输出时再进行一次encode，编码成我们需要的编码就可以了。

问题是，所有的字符都用一个int来表示会不会太浪费空间？毕竟，用ASCII编码，英文的字符只要一个字节就可以了。

确实会费点空间，但是现在的内存都足够大，而且我们只在程序内部使用这种方式，当字符串要写入文件或者通过网络传输时，我们都会进行相应的编码的。

还有一个问题，那些写死在程序中的字符串怎么办？难道每次使用都要进行一次decode?不同的操作系统默认使用的编码是不一样的，当我们在linux下，通常需要用utf8做decode，在Windows下，通常需要用gbk做 decode。这样，我们的代码就只能在特定的平台运行。

python给我们提供了一个很简单的办法，只要在字符串前面加一个u，它就会帮我们探测系统的编码，并自动完成decode。

6问题6：总结下，学到了什么?

本文用一个很常见的错误为起点，详细分析了python中的编码问题。我们看到了python处理字符问题的简单之处，也能够理解为什么python有这么强大的文本处理功能。

7测试题:看你是否真正理解了。

假设一台linux上有一个文件a.txt，里面的内容是"中文"两个字符，编码方式是utf-8。

现在，在python程序中写如下语句:

```
import codec
s=""
with codec.open("a.txt",encoding="utf-8") as f:
s=f.readline().strip()

with open("b.txt","w") as f:
f.write(s)
```

请问这段代码能执行吗？为什么?

答案:不能!

s底下的表示是unicode,写出时python会对其进行编码，默认用的ascii编码无法对"中文"两个字符进行编码，所以会报错!

作者：milter

链接：https://www.jianshu.com/p/eb22cee6c553