字符集

计算机只能处理数字(其实就是数字0和数字1)，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），0 - 255被用来表示大小写英文字母、数字和一些符号，这个编码表被称为ASCII编码。

ASCII码表使用7位二进制表示一个字符，它的区间范围时0~127，一共只能表示128个字符，仅能支持英语。随着计算机科学的发展，西欧语言、希腊语、泰语、阿拉伯语、希伯来语等语言的字符也被添加到码表中，形成了一个新的码表ISO8859-1(又被称为Latin1)码表。ISO8859-1使用8位二进制表示一个字符串，完全兼容ASCII码表。

Unicode（统一码、万国码、单一码）是计算机科学领域里的一项业界标准，包括字符集、编码方案等。Unicode 是为了解决传统的字符编码方案的局限而产生的，它为每种语言中的每个字符设定了统一并且唯一的二进制编码，以满足跨语言、跨平台进行文本转换、处理的要求。

字符和编码相互转换

使用chr和ord方法，可以实现字符和编码之间的相互转换。

```
print(ord('a'))  # 使用ord方法，可以获取一个字符对应的编码
print(chr(100))  # 使用chr方法，可以获取一个编码对应的字符
```

编码规则

使用Unicode为每种语言的每个字符都设定了唯一的二进制编码，但是它还是存在一定的问题，不够完美。

例如，汉字 “你” 转换成为一个字符结果是 `0x4f60` ,转换成为二进制就是 `01001111 01100000` ，此时就有两个问题:

1. `1001111 01100000` 到底是一个汉字 “你” ，还是两个 Latin1 字符?
2. 如果Unicode进行了规定，每个字符都使用n个八位来表示，对于Latin1字符来说，又会浪费很多存储空间。

为了解决这个问题，就出现了一些编码规则，按照一定的编码规则对Unicode数字进行计算，得出新的编码。在中国常用的字符编码有 `GBK` , `Big5` 和 `utf8` 这三种编码规则。

使用字符串的encode方法，可以将字符串按照指定的编码格式转换称为二进制；使用decode方法，可以将一个二进制数据按照指定的编码格式转换成为字符串。

```
s1 = '你'.encode('utf8')  # 将字符 你 按照utf8格式编码称为二进制
print(type(s1))  # <class 'bytes'>
print(s1)  # b'\xe4\xbd\xa0'

s2 = s1.decode('utf8') # 将二进制按照utf8格式解码称为字符串
print(s2)

s3 = '你'.encode('gbk') # 将字符 你 按照gbk格式转换称为二进制
print(s3)  # b'\xc4\xe3'

s4 = s3.decode('gbk')  # 将二进制按照gbk格式解码称为字符
print(s4)
```

思考：文字产生乱码的原因以及解决方案。