# 文本

# 1 字符编码系统

\*\*字符集（Charset）\*\*：是一个系统支持的所有抽象字符的集合。字符是各种文字和符号的总称，包括各国家文字、标点符号、图形符号、数字等。

\*\*字符编码（Character Encoding）\*\*：是字符集中字符的逻辑编码。字符集规定了所能显示的字符，而字符编码规定了字符存储时的规则。根据字符的编码信息，找到字符在字符集中的位置，显示出正确的字符。

\*\*文本文件\*\*：文本的显著特点是只能存储已知字符集中的字符。

\*\*二进制文件\*\*：二进制文件可以存储各种信息，信息被解码后不必是字符，可能是某种符号。二进制文件更复杂。

## 1.1 字符集

常见字符集名称：ASCII字符集、GB2312字符集、BIG5字符集、GB18030字符集、Unicode字符集等。

## 1.2 ASCII编码

\*\*ASCII\*\*（\*\*A\*\*merican \*\*S\*\*tandard \*\*C\*\*ode for \*\*I\*\*nformation \*\*I\*\*nterchange，\*\*美国信息交换标准代码\*\*）是基于[拉丁字母](http://zh.wikipedia.org/wiki/%E6%8B%89%E4%B8%81%E5%AD%97%E6%AF%8D)的一套[电脑](http://zh.wikipedia.org/wiki/%E7%94%B5%E8%84%91)[编码](http://zh.wikipedia.org/wiki/%E7%BC%96%E7%A0%81)系统。它主要用于显示[现代英语](http://zh.wikipedia.org/wiki/%E7%8F%BE%E4%BB%A3%E8%8B%B1%E8%AA%9E)，而其扩展版本EASCII则可以勉强显示其他[西欧](http://zh.wikipedia.org/wiki/%E8%A5%BF%E6%AC%A7)[语言](http://zh.wikipedia.org/wiki/%E8%AF%AD%E8%A8%80)。它是现今最通用的单[字节](http://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82)编码系统（但是有被Unicode追上的迹象），并等同于国际标准\*\*ISO/IEC 646\*\*。该套系统同时提供了字符集和字符编码

\*\*ASCII字符集\*\*：主要包括控制字符（回车键、退格、换行键等）；可显示字符（英文大小写字符、阿拉伯数字和西文符号）。

\*\*ASCII编码\*\*：将ASCII字符集转换为计算机可以接受的数字系统的数的规则。使用7位（bits）表示一个字符，共128字符；但是7位编码的字符集只能支持128个字符，为了表示更多的[欧洲](http://baike.baidu.com/view/3622.htm)常用字符对ASCII进行了扩展，ASCII扩展字符集使用8位（bits）表示一个字符，共256字符。ASCII字符集映射到数字编码规则如下图所示：

![img](https://github.com/wutongtongshu/doc/raw/master/%E8%A7%A3%E9%87%8A%E6%80%A7%E8%AF%AD%E8%A8%80/ASCll%E5%AD%97%E7%AC%A6%E9%9B%86.png)

图1 ASCII编码表

![img](https://github.com/wutongtongshu/doc/raw/master/%E8%A7%A3%E9%87%8A%E6%80%A7%E8%AF%AD%E8%A8%80/ASCll%E5%AD%97%E7%AC%A6%E9%9B%862.png)

图2 扩展ASCII编码表

ASCII的最大缺点是只能显示26个基本拉丁字母、阿拉伯数目字和英式标点符号，因此只能用于显示现代美国英语（而且在处理英语当中的外来词如naïve、café、élite等等时，所有重音符号都不得不去掉，即使这样做会违反拼写规则）。而EASCII虽然解决了部份西欧语言的显示问题，但对更多其他语言依然无能为力。因此现在的苹果电脑已经抛弃ASCII而转用[Unicode](http://zh.wikipedia.org/wiki/Unicode)。

## 1.3 GBXXXX编码方案

天朝专家把那些127号之后的奇异符号们（即EASCII）取消掉，规定：一个小于127的字符的意义与原来相同，但两个大于127的字符连在一起时，就表示一个汉字，前面的一个字节（他称之为高字节）从0xA1用到 0xF7，后面一个字节（低字节）从0xA1到0xFE，这样我们就可以组合出大约7000多个简体汉字了。在这些编码里，还把数学符号、罗马希腊的 字母、日文的假名们都编进去了，连在ASCII里本来就有的数字、标点、字母都统统重新编了两个字节长的编码，这就是常说的"全角"字符，而原来在127号以下的那些就叫"半角"字符了。

上述编码规则就是\*\*GB2312\*\*。GB 2312只收录6763个汉字，GBK是对GB2312的扩展，也就是[\*CP936\*](http://zh.wikipedia.org/w/index.php?title=CP936&action=edit&redlink=1)\*字码表 GBK自身并非国家标准，只是曾由[国家技术监督局](http://zh.wikipedia.org/wiki/%E5%9B%BD%E5%AE%B6%E6%8A%80%E6%9C%AF%E7%9B%91%E7%9D%A3%E5%B1%80)标准化司、[电子工业部](http://zh.wikipedia.org/wiki/%E7%94%B5%E5%AD%90%E5%B7%A5%E4%B8%9A%E9%83%A8)科技与质量监督司公布为"技术规范指导性文件"。国家标准[GB18030](http://zh.wikipedia.org/wiki/GB18030)技术上兼容GBK而非GB13000。

\*\*GB 18030\*\*，全称：[国家标准](http://zh.wikipedia.org/wiki/%E4%B8%AD%E5%8D%8E%E4%BA%BA%E6%B0%91%E5%85%B1%E5%92%8C%E5%9B%BD%E5%9B%BD%E5%AE%B6%E6%A0%87%E5%87%86)GB 18030-2005《信息技术 中文编码字符集》，是[中华人民共和国](http://zh.wikipedia.org/wiki/%E4%B8%AD%E5%8D%8E%E4%BA%BA%E6%B0%91%E5%85%B1%E5%92%8C%E5%9B%BD)现时最新的内码字集，是GB 18030-2000《信息技术 信息交换用汉字编码字符集 基本集的扩充》的修订版。与[GB 2312-1980](http://zh.wikipedia.org/wiki/GB\_2312)完全兼容，与[GBK](http://zh.wikipedia.org/wiki/GBK)基本兼容，支持[GB 13000](http://zh.wikipedia.org/wiki/GB\_13000)及[Unicode](http://zh.wikipedia.org/wiki/Unicode)的全部统一汉字，共收录汉字70244个。GB 18030主要有以下特点：

- 与[UTF-8](http://zh.wikipedia.org/wiki/UTF-8)相同，采用多[字节](http://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82)编码，每个字可以由1个、2个或4个字节组成。

- 编码空间庞大，最多可定义161万个字符。

- 支持中国国内[少数民族](http://zh.wikipedia.org/wiki/%E5%B0%91%E6%95%B0%E6%B0%91%E6%97%8F)的文字，不需要动用造字区。

- 汉字收录范围包含繁体汉字以及日韩汉字

![img](https://github.com/wutongtongshu/doc/raw/master/%E8%A7%A3%E9%87%8A%E6%80%A7%E8%AF%AD%E8%A8%80/GB2312%E9%83%A8%E5%88%86.png)

图4 GB18030编码总体结构

## 1.4 BIG5字符集&编码

\*\*Big5\*\*，又称为\*\*大五码\*\*或\*\*五大码\*\*，是使用[繁体中文](http://zh.wikipedia.org/wiki/%E7%B9%81%E4%BD%93%E4%B8%AD%E6%96%87)（正体中文）社区中最常用的电脑[汉字](http://zh.wikipedia.org/wiki/%E6%B1%89%E5%AD%97)[字符集](http://zh.wikipedia.org/wiki/%E5%AD%97%E7%AC%A6%E9%9B%86)标准，共收录13,060个汉字。

## 1.5 伟大的创想Unicode

GB232/GBK/GB18030/BIG5各搞一套，在本地使用没有问题，一旦出现在网络中，由于不兼容，互相访问就出现了乱码现象。为了解决这个问题，一个伟大的创想产生了——Unicode。Unicode编码系统为表达任意语言的任意字符而设计。它使用4字节的数字来表达每个字母、符号，或者表意文字(ideograph)。每个数字代表唯一的至少在某种语言中使用的符号。\*\*Unicode\*\*（\*\*统一码\*\*、\*\*万国码\*\*、\*\*单一码\*\*、\*\*标准万国码\*\*）是业界的一种标准，它可以使电脑得以体现世界上数十种文字的系统。Unicode 是基于[通用字符集](http://zh.wikipedia.org/wiki/%E9%80%9A%E7%94%A8%E5%AD%97%E7%AC%A6%E9%9B%86)（Universal Character Set）的标准来发展，并且同时也以书本的形式[[1\]](http://zh.wikipedia.org/wiki/Unicode)对外发表。Unicode 还不断在扩增， 每个新版本插入更多新的字符。直至目前为止的第六版，Unicode 就已经包含了超过十万个[字符](http://zh.wikipedia.org/wiki/%E5%AD%97%E7%AC%A6)（在[2005年](http://zh.wikipedia.org/wiki/2005%E5%B9%B4)，Unicode 的第十万个字符被采纳且认可成为标准之一）、一组可用以作为视觉参考的代码图表、一套编码方法与一组标准[字符编码](http://zh.wikipedia.org/wiki/%E5%AD%97%E7%AC%A6%E7%BC%96%E7%A0%81)、一套包含了上标字、下标字等字符特性的枚举等。Unicode 组织（The Unicode Consortium）是由一个非营利性的机构所运作，并主导 Unicode 的后续发展，其目标在于：将既有的字符编码方案以Unicode 编码方案来加以取代，特别是既有的方案在多语环境下，皆仅有有限的空间以及不兼容的问题。

### 1.5.1 Unicode字符集的三种编码

UTF-32

上述使用4字节的数字来表达每个字母、符号，或者表意文字(ideograph)，每个数字代表唯一的至少在某种语言中使用的符号的编码方案，称为UTF-32。UTF-32又称\*\*UCS-4\*\*是一种将[Unicode](http://zh.wikipedia.org/wiki/Unicode)字符编码的协定，对每个字符都使用4字节。就空间而言，是非常没有效率的。

这种方法有其优点，最重要的一点就是可以在常数时间内定位字符串里的第N个字符，因为第N个字符从第4×Nth个字节开始。虽然每一个码位使用固定长定的字节看似方便，它并不如其它Unicode编码使用得广泛。

UTF-16

尽管有Unicode字符非常多，但是实际上大多数人不会用到超过前65535个以外的字符。因此，就有了另外一种Unicode编码方式，叫做UTF-16(因为16位 = 2字节)。UTF-16将0–65535范围内的字符编码成2个字节，如果真的需要表达那些很少使用的"星芒层(astral plane)"内超过这65535范围的Unicode字符，则需要使用一些诡异的技巧来实现。UTF-16编码最明显的优点是它在空间效率上比UTF-32高两倍，因为每个字符只需要2个字节来存储（除去65535范围以外的），而不是UTF-32中的4个字节。并且，如果我们假设某个字符串不包含任何星芒层中的字符，那么我们依然可以在常数时间内找到其中的第N个字符，直到它不成立为止这总是一个不错的推断。其编码方法是：

- 如果字符编码U小于0x10000，也就是十进制的0到65535之内，则直接使用两字节表示；

- 如果字符编码U大于0x10000，由于UNICODE编码范围最大为0x10FFFF，从0x10000到0x10FFFF之间 共有0xFFFFF个编码，也就是需要20个bit就可以标示这些编码。用U'表示从0-0xFFFFF之间的值，将其前 10 bit作为高位和16 bit的数值0xD800进行 逻辑or 操作，将后10 bit作为低位和0xDC00做 逻辑or 操作，这样组成的 4个byte就构成了U的编码。

对于UTF-32和UTF-16编码方式还有一些其他不明显的缺点。不同的计算机系统会以不同的顺序保存字节。这意味着字符U+4E2D在UTF-16编码方式下可能被保存为4E 2D或者2D 4E，这取决于该系统使用的是大尾端(big-endian)还是小尾端(little-endian)。（对于UTF-32编码方式，则有更多种可能的字节排列。）只要文档没有离开你的计算机，它还是安全的——同一台电脑上的不同程序使用相同的字节顺序(byte order)。但是当我们需要在系统之间传输这个文档的时候，也许在万维网中，我们就需要一种方法来指示当前我们的字节是怎样存储的。不然的话，接收文档的计算机就无法知道这两个字节4E 2D表达的到底是U+4E2D还是U+2D4E。

为了解决这个问题，多字节的Unicode编码方式定义了一个"字节顺序标记(Byte Order Mark)"，它是一个特殊的非打印字符，你可以把它包含在文档的开头来指示你所使用的字节顺序。对于UTF-16，字节顺序标记是U+FEFF。如果收到一个以字节FF FE开头的UTF-16编码的文档，你就能确定它的字节顺序是单向的(one way)的了；如果它以FE FF开头，则可以确定字节顺序反向了。

UTF-8

\*\*UTF-8\*\*（8-bit Unicode Transformation Format）是一种针对[Unicode](http://zh.wikipedia.org/wiki/Unicode)的可变长度[字符编码](http://zh.wikipedia.org/wiki/%E5%AD%97%E7%AC%A6%E7%BC%96%E7%A0%81)（[定长码](http://zh.wikipedia.org/w/index.php?title=%E5%AE%9A%E9%95%BF%E7%A0%81&action=edit&redlink=1)），也是一种[前缀码](http://zh.wikipedia.org/w/index.php?title=%E5%89%8D%E7%BC%80%E7%A0%81&action=edit&redlink=1)。它可以用来表示Unicode标准中的任何字符，且其编码中的第一个[字节](http://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82)仍与[ASCII](http://zh.wikipedia.org/wiki/ASCII)兼容，这使得原来处理ASCII字符的[软件](http://zh.wikipedia.org/wiki/%E8%BB%9F%E9%AB%94)无须或只须做少部份修改，即可继续使用。因此，它逐渐成为[电子邮件](http://zh.wikipedia.org/wiki/%E9%9B%BB%E5%AD%90%E9%83%B5%E4%BB%B6)、[网页](http://zh.wikipedia.org/wiki/%E7%B6%B2%E9%A0%81)及其他[存储](http://zh.wikipedia.org/wiki/%E5%84%B2%E5%AD%98%E8%A3%9D%E7%BD%AE)或传送文字的应用中，优先采用的编码。[互联网工程工作小组](http://zh.wikipedia.org/wiki/%E7%B6%B2%E9%9A%9B%E7%B6%B2%E8%B7%AF%E5%B7%A5%E7%A8%8B%E5%B7%A5%E4%BD%9C%E5%B0%8F%E7%B5%84)（IETF）要求所有[互联网](http://zh.wikipedia.org/wiki/%E7%B6%B2%E9%9A%9B%E7%B6%B2%E8%B7%AF)[协议](http://zh.wikipedia.org/wiki/%E7%BD%91%E7%BB%9C%E5%8D%8F%E8%AE%AE)都必须支持UTF-8编码。

UTF-8使用一至四个[字节](http://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82)为每个字符编码：

1. 128个US-ASCII字符只需一个字节编码（Unicode范围由U+0000至U+007F）。

2. 带有[附加符号](http://zh.wikipedia.org/wiki/%E9%99%84%E5%8A%A0%E7%AC%A6%E5%8F%B7)的[拉丁文](http://zh.wikipedia.org/wiki/%E6%8B%89%E4%B8%81%E6%96%87)、[希腊文](http://zh.wikipedia.org/wiki/%E5%B8%8C%E8%87%98%E6%96%87)、[西里尔字母](http://zh.wikipedia.org/wiki/%E8%A5%BF%E9%87%8C%E7%88%BE%E5%AD%97%E6%AF%8D)、[亚美尼亚语](http://zh.wikipedia.org/wiki/%E4%BA%9E%E7%BE%8E%E5%B0%BC%E4%BA%9E%E8%AA%9E)、[希伯来文](http://zh.wikipedia.org/wiki/%E5%B8%8C%E4%BC%AF%E4%BE%86%E6%96%87)、[阿拉伯文](http://zh.wikipedia.org/wiki/%E9%98%BF%E6%8B%89%E4%BC%AF%E6%96%87)、[叙利亚文](http://zh.wikipedia.org/wiki/%E5%8F%99%E5%88%A9%E4%BA%9A%E6%96%87)及[它拿字母](http://zh.wikipedia.org/wiki/%E5%AE%83%E6%8B%BF%E5%AD%97%E6%AF%8D)则需要二个字节编码（Unicode范围由U+0080至U+07FF）。

3. 其他[基本多文种平面](http://zh.wikipedia.org/wiki/%E5%9F%BA%E6%9C%AC%E5%A4%9A%E6%96%87%E7%A8%AE%E5%B9%B3%E9%9D%A2)（BMP）中的字符（这包含了大部分常用字）使用三个字节编码。

4. 其他极少使用的Unicode[辅助平面](http://zh.wikipedia.org/wiki/%E8%BC%94%E5%8A%A9%E5%B9%B3%E9%9D%A2)的字符使用四字节编码。

在处理经常会用到的ASCII字符方面非常有效。在处理扩展的拉丁字符集方面也不比UTF-16差。对于中文字符来说，比UTF-32要好。同时，（在这一条上你得相信我，因为我不打算给你展示它的数学原理。）由位操作的天性使然，使用UTF-8不再存在字节顺序的问题了。一份以utf-8编码的文档在不同的计算机之间是一样的比特流。

总体来说，在Unicode字符串中不可能由码点数量决定显示它所需要的长度，或者显示字符串之后在文本缓冲区中光标应该放置的位置；组合字符、变宽字体、不可打印字符和从右至左的文字都是其归因。所以尽管在UTF-8字符串中字符数量与码点数量的关系比UTF-32更为复杂，在实际中很少会遇到有不同的情形。

UTF-8是一种变长字节编码方式。对于某一个字符的UTF-8编码，如果只有一个字节则其最高二进制位为0；如果是多字节，其第一个字节从最高位开始，连续的二进制位值为1的个数决定了其编码的位数，其余各字节均以10开头。UTF-8最多可用到6个字节。

所以，如果一个字符编码成3个字节，但是一个字节数组的结尾可能只包含了其中两个字节，而后一个字节数组开头包含了该字符编码的最后一个字节，那么，如果两个字节数组单独解码，就会发生乱码。

要解决这个问题，要了解UTF-8的编码规则，如下所示：

1字节0xxxxxxx

2字节110xxxxx 10xxxxxx

3字节1110xxxx 10xxxxxx 10xxxxxx

4字节11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

5字节111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

6字节1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

知道了编码规则，就知道如何处理这种情况了

当要对一个字节数组进行解码时，要对这个字节数组最后几个字节进行判断，如果是一个完整的字符，则可以进行解码

否则，要将不完整的字节截取出来，拼接到下一字节数组后进行解码。

关于UTF-8占几个字节的问题

占2个字节的：〇

占3个字节的：基本等同于GBK，含21000多个汉字

占4个字节的：中日韩超大字符集里面的汉字，有5万多个

一个utf8数字占1个字节

一个utf8英文字母占1个字节

在查找 UTF-8 编码资料时发现，很多的帖子说的 UTF-8 编码里，一个汉字占用3个字节，有的还做了个证明，大概是这样的，创建一个没有BOM的UTF-8编码的文本文件，里面保存了几个汉字，然后查看文件的大小。我觉得这样的证明没有一点说服力，因为 UTF-8 是变长的，1-6个字节，少量的汉字检测是不能说明所有的汉字都是的。

后来我又查看了字符映射表－汉语，找到了正确的答案，少数是汉字每个占用3个字节，多数占用4个字节。

占用3个字节的范围

[text] view plaincopy

U+2E80 - U+2EF3 : 0xE2 0xBA 0x80 - 0xE2 0xBB 0xB3 共 115 个

U+2F00 - U+2FD5 : 0xE2 0xBC 0x80 - 0xE2 0xBF 0x95 共 213 个

U+3005 - U+3029 : 0xE3 0x80 0x85 - 0xE3 0x80 0xA9 共 36 个

U+3038 - U+4DB5 : 0xE3 0x80 0xB8 - 0xE4 0xB6 0xB5 共 7549 个

U+4E00 - U+FA6A : 0xE4 0xB8 0x80 - 0xEF 0xA9 0xAA 共 44138 个

U+FA70 - U+FAD9 : 0xEF 0xA9 0xB0 - 0xEF 0xAB 0x99 共 105 个

合计： 52156 个

\*\*占用4个字节的范围\*\*

[text] view plaincopy

U+20000 - U+2FA1D : 0xF0 0xA0 0x80 0x80 - 0xF0 0xAF 0xA8 0x9D 共 64029 个

合计： 64029 个

\*\*优点\*\*

- UTF-8是ASCII的一个[超集](http://zh.wikipedia.org/wiki/%E8%B6%85%E9%9B%86)。因为一个纯ASCII字符串也是一个合法的UTF-8字符串，所以现存的ASCII文本不需要转换。为传统的扩展ASCII字符集设计的软件通常可以不经修改或很少修改就能与UTF-8一起使用。

- 使用标准的面向字节的排序例程对UTF-8排序将产生与基于Unicode代码点排序相同的结果。（尽管这只有有限的有用性，因为在任何特定语言或文化下都不太可能有仍可接受的文字排列顺序。）

- UTF-8和UTF-16都是[可扩展标记语言](http://zh.wikipedia.org/wiki/%E5%8F%AF%E6%89%A9%E5%B1%95%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80)文档的标准编码。所有其它编码都必须通过显式或文本声明来指定。

- 任何[面向字节](http://zh.wikipedia.org/w/index.php?title=%E9%9D%A2%E5%90%91%E5%AD%97%E8%8A%82&action=edit&redlink=1)的[字符串搜索算法](http://zh.wikipedia.org/wiki/%E5%AD%97%E7%AC%A6%E4%B8%B2%E6%90%9C%E7%B4%A2%E7%AE%97%E6%B3%95)都可以用于UTF-8的数据（只要输入仅由完整的UTF-8字符组成）。但是，对于包含字符记数的正则表达式或其它结构必须小心。

- UTF-8字符串可以由一个简单的算法可靠地识别出来。就是，一个字符串在任何其它编码中表现为合法的UTF-8的可能性很低，并随字符串长度增长而减小。举例说，字符值C0,C1,F5至FF从来没有出现。为了更好的可靠性，可以使用正则表达式来统计非法过长和替代值（可以查看[W3 FAQ: Multilingual Forms](http://www.w3.org/International/questions/qa-forms-utf-8)上的验证UTF-8字符串的正则表达式）。

\*\*缺点\*\*

因为每个字符使用不同数量的字节编码，所以寻找串中第N个字符是一个O(N)复杂度的操作 — 即，串越长，则需要更多的时间来定位特定的字符。同时，还需要位变换来把字符编码成字节，把字节解码成字符。

UCS & UNICODE

\*\*通用字符集\*\*（Universal Character Set，\*\*UCS\*\*）是由[ISO](http://zh.wikipedia.org/wiki/%E5%9C%8B%E9%9A%9B%E6%A8%99%E6%BA%96%E5%8C%96%E7%B5%84%E7%B9%94)制定的\*\*ISO 10646\*\*（或称\*\*ISO/IEC 10646\*\*）标准所定义的标准字符集。历史上存在两个独立的尝试创立单一字符集的组织，即国际标准化组织（ISO）和多语言软件制造商组成的[统一码联盟](http://zh.wikipedia.org/wiki/%E7%B5%B1%E4%B8%80%E7%A2%BC%E8%81%AF%E7%9B%9F)。前者开发的 ISO/IEC 10646 项目，后者开发的[统一码](http://zh.wikipedia.org/wiki/%E7%B5%B1%E4%B8%80%E7%A2%BC)项目。因此最初制定了不同的标准。

[1991年](http://zh.wikipedia.org/wiki/1991%E5%B9%B4)前后，两个项目的参与者都认识到，世界不需要两个不兼容的字符集。于是，它们开始合并双方的工作成果，并为创立一个单一编码表而协同工作。从Unicode 2.0开始，Unicode采用了与ISO 10646-1相同的字库和字码；ISO也承诺，ISO 10646将不会替超出U+10FFFF的UCS-4编码赋值，以使得两者保持一致。两个项目仍都存在，并独立地公布各自的标准。但统一码联盟和ISO/IEC JTC1/SC2都同意保持两者标准的码表兼容，并紧密地共同调整任何未来的扩展。在发布的时候，Unicode一般都会采用有关字码最常见的字型，但ISO 10646一般都尽可能采用[Century字型](http://zh.wikipedia.org/w/index.php?title=Century%E5%AD%97%E5%9E%8B&action=edit&redlink=1)。

## 1.4 浏览器头部参数

在HTTP中，与字符集和字符编码相关的消息头是Accept-Charset/Content-Type，另外主区区分Accept-Charset/Accept-Encoding/Accept-Language/Content-Type/Content-Encoding/Content-Language：

Accept-Charset：浏览器申明自己接收的字符集，这就是本文前面介绍的各种字符集和字符编码，如gb2312，utf-8（通常我们说Charset包括了相应的字符编码方案）；

Accept-Encoding：浏览器申明自己接收的编码方法，通常指定压缩方法，是否支持压缩，支持什么压缩方法（gzip，deflate），（注意：这不是只字符编码）；

Accept-Language：浏览器申明自己接收的语言。语言跟字符集的区别：中文是语言，中文有多种字符集，比如big5，gb2312，gbk等等；

Content-Type：WEB服务器告诉浏览器自己响应的对象的类型和字符集。例如：Content-Type: text/html; charset='gb2312'

Content-Encoding：WEB服务器表明自己使用了什么压缩方法（gzip，deflate）压缩响应中的对象。例如：Content-Encoding：gzip

Content-Language：WEB服务器告诉浏览器自己响应的对象的语言。

# 2 html

## 2.1 常用标签

### 2.1.1 特殊标签

| <code> | 定义计算机代码文本 |

| ------ | ------------------ |

| <kbd> | 定义键盘文本 |

| <samp> | 定义计算机代码示例 |

| <var> | 定义变量 |

| <pre> | 定义预格式化文本 |

### 2.1.2 注释标签

属于网页的注释标签只有唯一的一种

<!-- -->

### 2.1.2 三种样式表

- 外部

```html

<head>

<link rel="stylesheet" type="text/css" href="mystyle.css">

</head>

```

- 内部

```html

<head>

<style type="text/css">

body {background-color: red}

p {margin-left: 20px}

</style>

</head>

```

- 内联

```html

<p style="color: red; margin-left: 20px">This is a paragraph</p>

```

### 2.1.3 html实体

第三列是实体简写，第四列是实体字符的ascll编码，两种都能解析。

| | 空格 | & nbsp; | &#160; |

| ---- | ----------------- | ----------------- | ------- |

| | | | &#60; |

| > | 大于号 | &gt; | &#62; |

| & | 和号 | &amp; | &#38; |

| " | 引号 | &quot; | &#34; |

| ' | 撇号 | &apos; (IE不支持) | &#39; |

| ￠ | 分（cent） | &cent; | &#162; |

| £ | 镑（pound） | &pound; | &#163; |

| ¥ | 元（yen） | &yen; | &#165; |

| € | 欧元（euro） | &euro; | &#8364; |

| § | 小节 | &sect; | &#167; |

| © | 版权（copyright） | &copy; | &#169; |

| ® | 注册商标 | &reg; | &#174; |

| ™ | 商标 | &trade; | &#8482; |

| × | 乘号 | &times; | &#215; |

| ÷ | 除号 | &divide; | &#247; |

### 2.1.4 URL 编码

URL编码跟页面内容没有任何关系。URL 只能使用 [ASCII 字符集](http://www.w3school.com.cn/tags/html\_ref\_ascii.asp)来通过因特网进行发送。由于 URL 常常会包含 ASCII 集合之外的字符，URL 必须转换为有效的 ASCII 格式。所以在发送非ASCll字符时，需要我们将URL处理好之后，传出去。所以，如果在url中使用了+号，要小心。

URL 编码使用 "%" 其后跟随两位的十六进制数来替换非 ASCII 字符。

URL 不能包含空格。URL 编码通常使用 + 来替换空格。

不会被编码的内容

1.大写字母A-Z

2.小写字母a-z

3.数字 0-9

4.标点符 - \_ . ! ~ \* ' (和 ,)

注意，浏览器一般会自动使用上述编码方案，URLEncoder跟这个类很像，总之就是按照上面的来。

[url编码对照表](http://www.w3school.com.cn/tags/html\_ref\_urlencode.html)，注意编码方案并不是把所有字符都编码，这个对照表只是起对照作用。

# 3 操作

## 3.1 上传文件

[5种方式](https://www.cnblogs.com/sunliyuan/p/5737928.html) 可实现文件上传到服务器，通过表单的action动作，指明处理程序（可以是某个 url ），对上传文件进行后续处理。

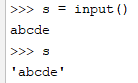
[处理文件上传](https://www.cnblogs.com/Java3y/p/8428591.html)

# Python

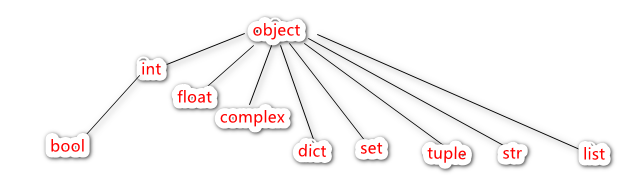
## 系统api

### Input()

标准输入



## 基础类型



a = **1**;  
b = **1.1**;  
c = **1** + **2j**;  
d = **True**;  
e = **None**;  
set1 = {**1**, **2**, **3**};  
lst1 = [**1**, **2**, **4**];  
dict = {**"x"**:**1**, **"y"**:**2**}  
tuple = (**1**, **2**, **3**, **4**)

str = “sss”

## 字面量

### 数字

数字大小没限制

a = 10;

a = 0b0101; 二进制数

a = 0o10; 八进制数

a = 0x111; 十六进制数

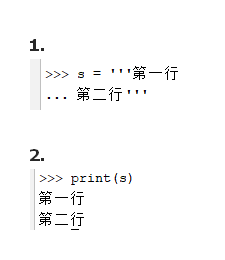
### 字符串

支持单双引号，单引号双引号相同，不可以同种引号嵌套，可以相互包含，不可以包含换行符。若换行用 \

S = ‘hello’;

S = “hello”

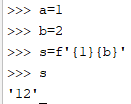
换行使用 ‘’’或者 “”” 三重引号，这样字符串格式全部保留。



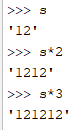
格式输出



插值器 f



重复输出

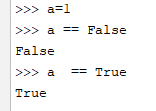


转义字符：

要么用 \ 作为转义字符，要么用 \uxxxx 这里是 unicode (utf-16) 编码。

### 布尔值

True/False 也是 1 和 0



### 空值

None 表示不存在，也是一种类型

## 类型检查

type() 函数做类型检查

isinstanceof

## 对象

对象有 id 和 type 属性。字面量只有一个。

## 比较运算符

== 比较值

Is/is not 比较地址

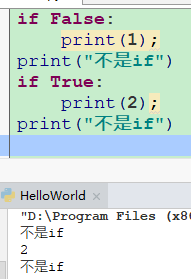
## 逻辑运算符

and not or

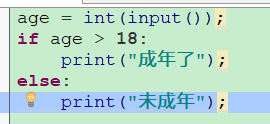
## 流程控制

### If

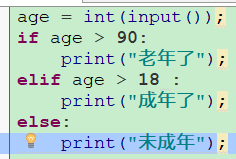
If 语句，python 以缩进标识代码块，要么用 tab 键，要么用四个空格。推荐用四个空格



If else



If elif else



## 函数

### 定义函数

def functionname(参数1，参数2):

"函数说明"

函数实现

return 返回表达式

fun = lamda x,y : x + y;

### 默认参数

def functionname(参数1，参数2 = “ ”):

"函数说明"

函数实现

return 返回表达式

### 不定参数

def functionname(x, \* sdkll):

"函数说明"

函数实现

return 返回表达式

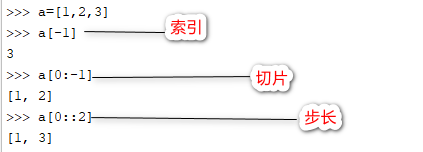
## 数据结构

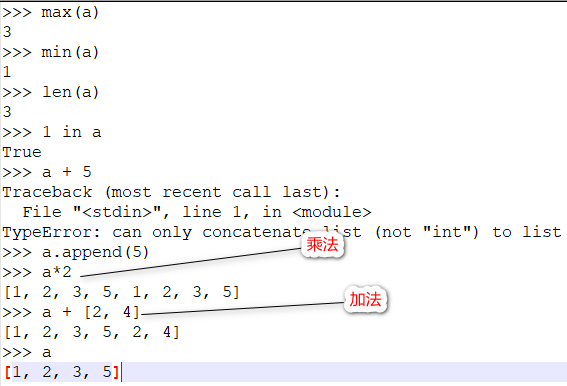
### 序列

序列是Python中最基本的数据结构。序列中的每个元素都分配一个数字 - 它的位置，或索引，第一个索引是0，第二个索引是1，依此类推。每个索引对应一个元素。Python包含 6 中内建的序列，包括list、tuple、string、Unicode字符串、buffer对象和xrange对象。对于序列，都可以使用以下操作：

索引、切片、加、乘、成员检查、计算序列的长度、取序列中的最大、最小值，这里类似于数组。

#### List





# 类

## 构造、继承

**class** Car:  
 '汽车模型'  
 **def** \_\_init\_\_(self, *mark*, *model*, *year*):  
 self.mark = *mark*;  
 self.model = *model*;  
 self.year = *year*;  
 self.odometer\_reading = **0**;  
  
 **def** \_\_init\_\_(self, *mark*):  
 self.mark = *mark*;  
  
 **def** get\_descriptive\_name(self):  
 long\_name = str(self.year) + **' '** + \  
 str(self.mark) + **' '** + str(self.model);  
 **return** long\_name;  
  
**class** ElectricCar(Car):  
 **def** \_\_init\_\_(self, *mark*):  
 super().\_\_init\_\_(*mark*, **None**, **None**);  
  
 **def** get\_descriptive\_name(self):  
 **return "子类重写"**

# 模块

import module1[, module2[,... moduleN]]

使用时要用全名

from modname import name1[, name2[, ... nameN]]

使用时用名字就行了

from modname import \*

全部导入，使用时用名字就行了

# 异常处理

try:

<语句> #运行别的代码

except <名字>：

<语句> #如果在try部份引发了'name'异常

except <名字>，<数据>:

<语句> #如果引发了'name'异常，获得附加的数据

else:

<语句> #如果没有异常发生