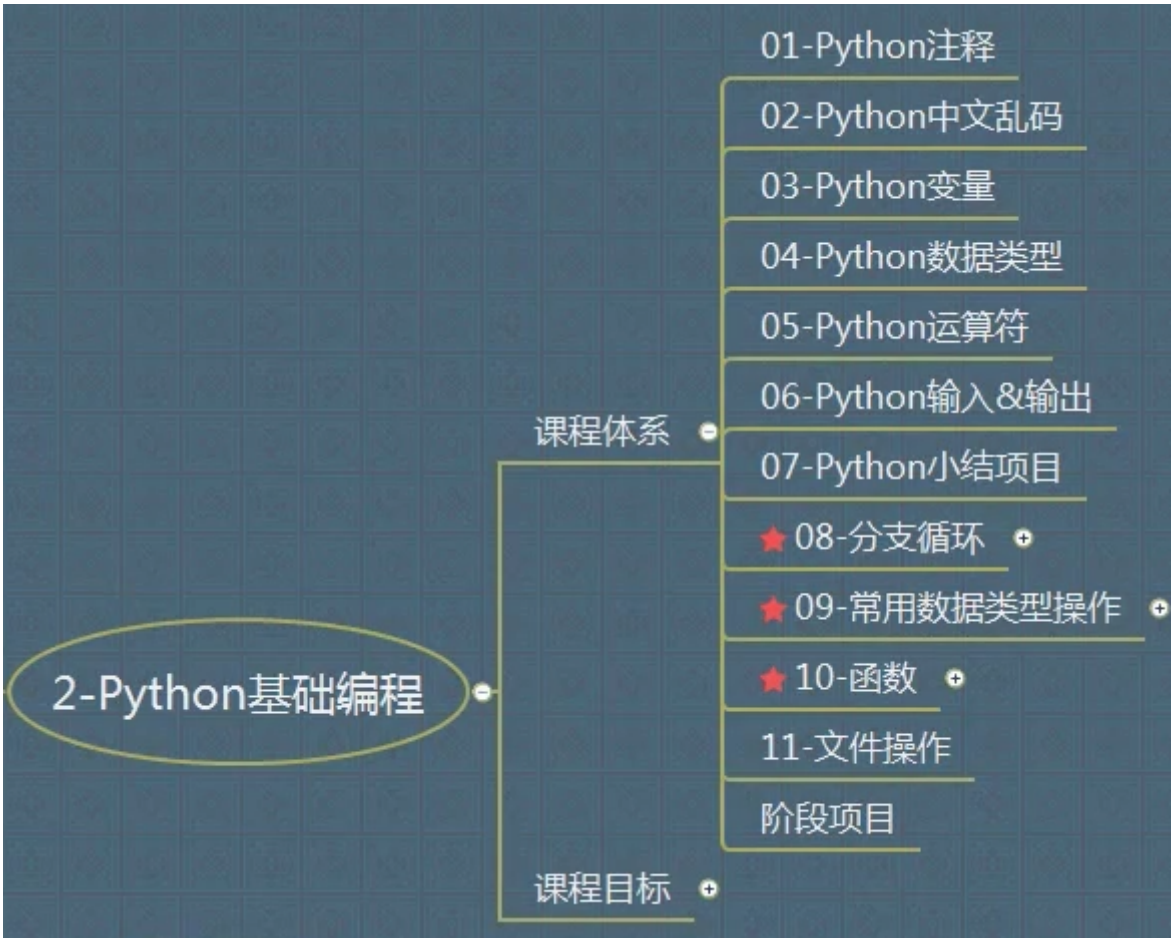
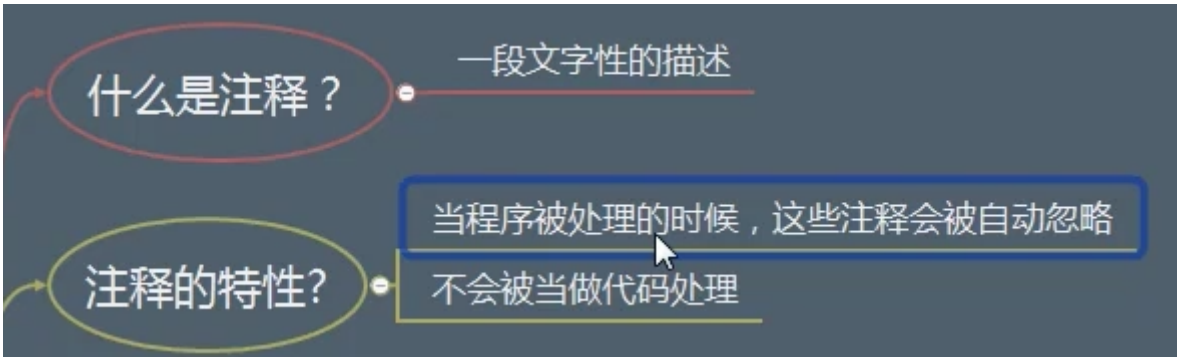
[笔记][LIKE-Python-2][02]

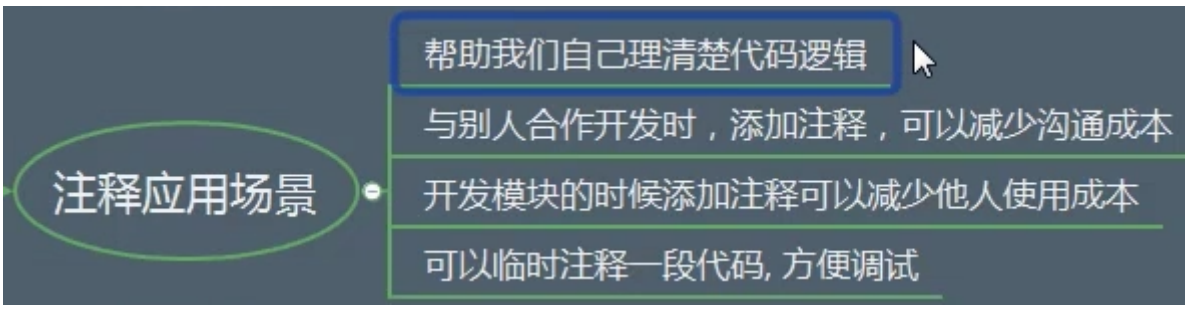
Python

001. 课程体系

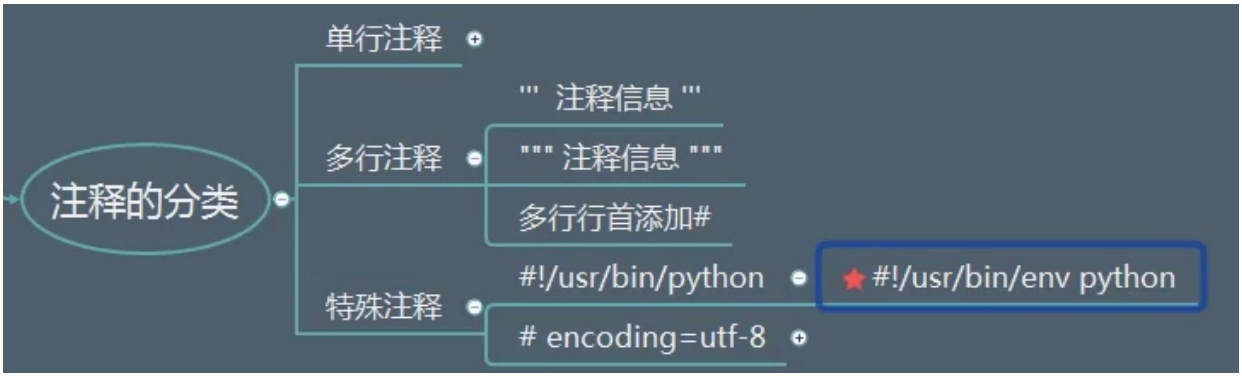


002. Python的注释





> 【!】`Ubuntu` 小技巧：把文件夹拖动到终端窗口，会自动复制文件夹的路径！



特殊注释

`Linux` 下可以这样执行一个文件 `./文件名`

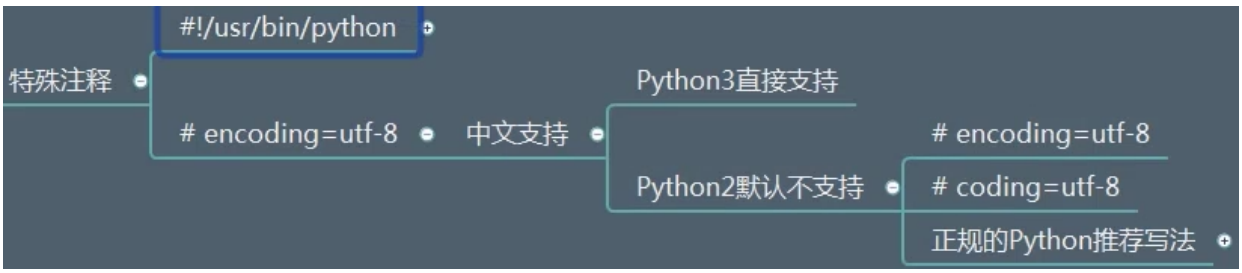
但是要赋予可执行权限 `chmod a+x a.py`

查看 `python` 解释器位置：`which python`

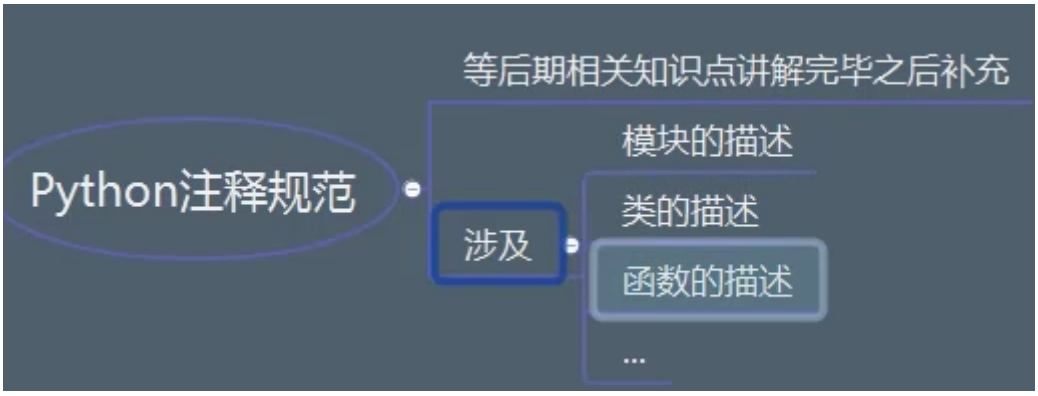
然后写在 `*.py` 文件第一行：`#!/usr/bin/python`

这样就可直接 `./` 运行了。

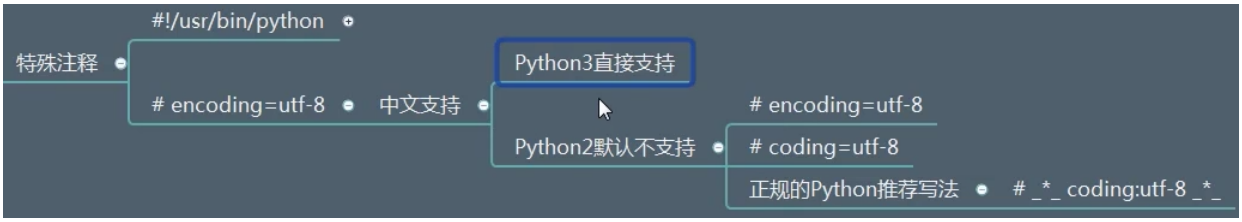
推荐这么写：`#!/usr/bin/env python`



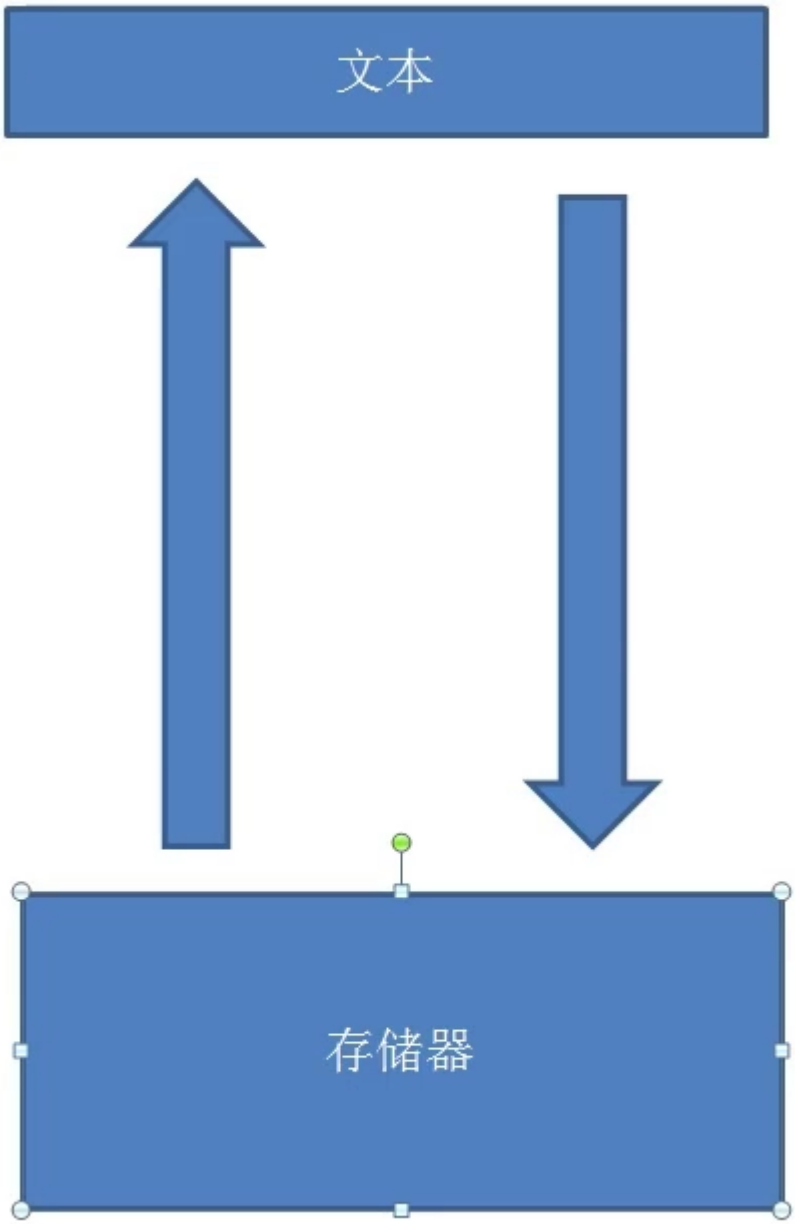
> **正规写法**
>
> `# -*- coding: utf-8 -*-`



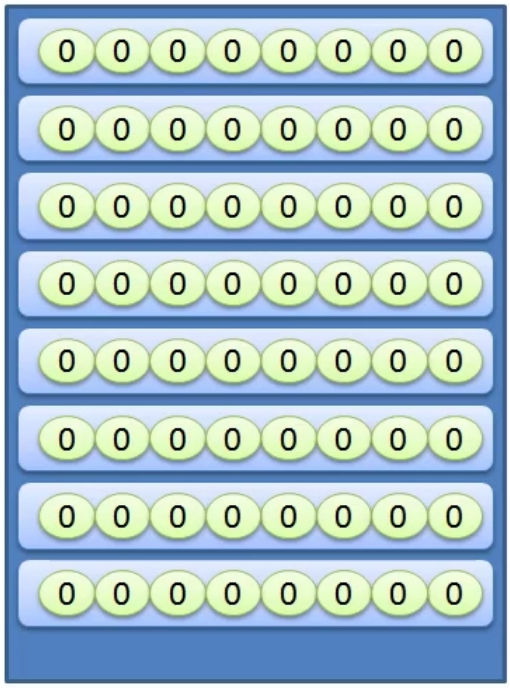
003. Python的中文乱码

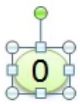


最终的目标：把文本存储到存储器


文本
存储器

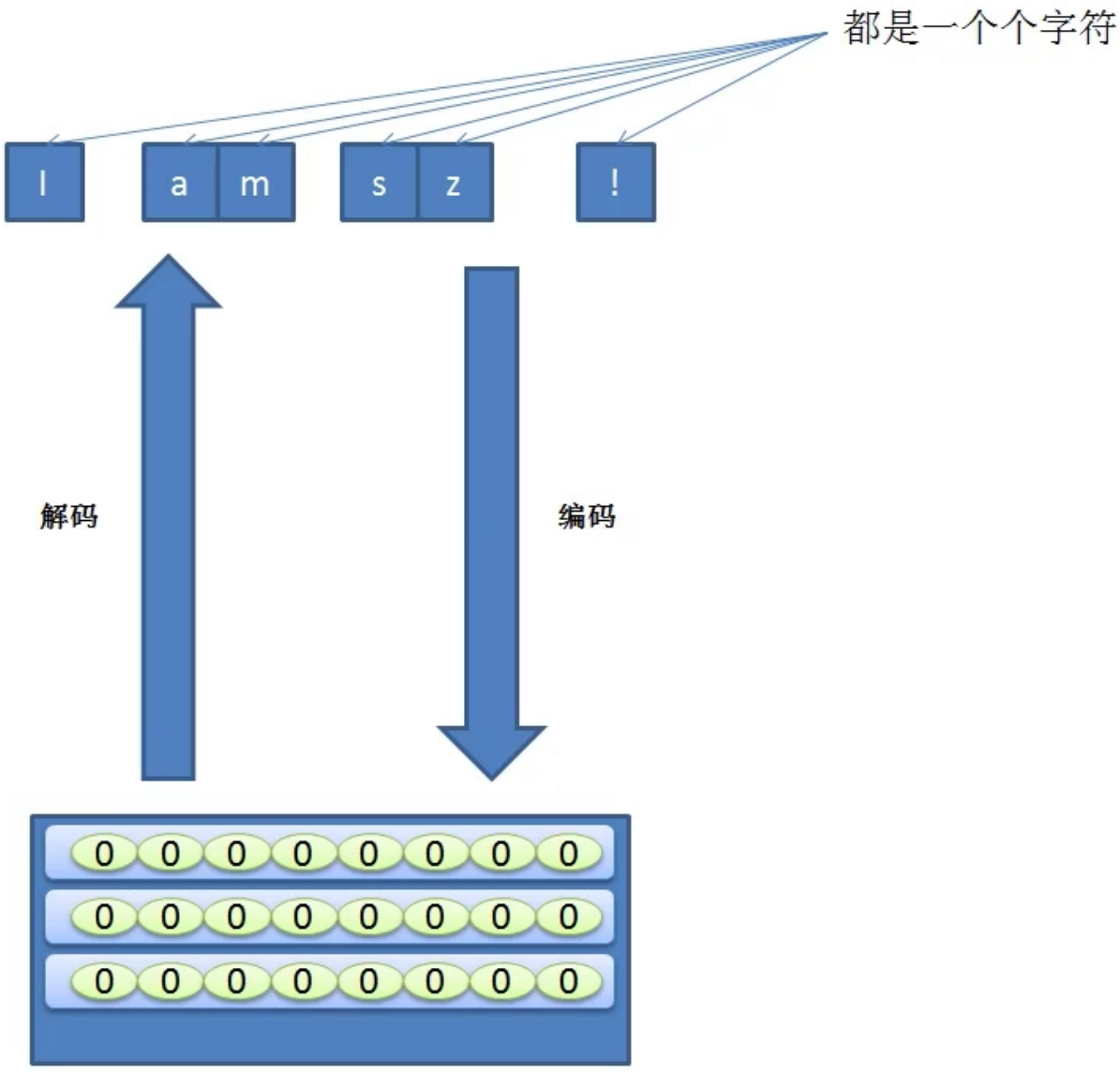
计算机存储器





一个比特位 -> 计算机最小存储单元

一个字节 -> 一个基本存储单元



如何存储字符？

- 先把字符转换成数字
- 然后直接存储数字

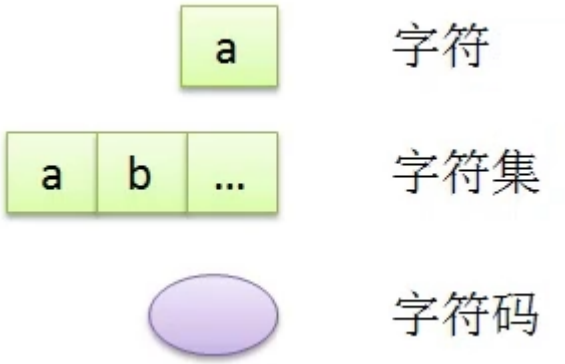
两个问题

1. 按照怎样的规范把字符转换成数字？
2. 按照怎样的方式存储数字（如：用多少个字节）？

字符编码
有两个方面，对应刚才的两个问题。

- 存储规范
- 存储方式

以上两个任意一个没有选择正确，解码的时候就会出现乱码。



所有字符的集合叫做**字符集**。
字符对应的数字叫做**字符码**。

为啥会有这么多字符编码?

- 1. 计算机最早发明于美国, 英语中主要就包括26个字母(大小写), 10个数字, 标点符号, 控制符等等; 所以, 最终就制定了ASCII字符编码, 映射了字符和字符码的关系, 并使用一个字节的后七位(0 - 127)来存储; (当时真TM没想到, 计算机会普及)

- 2. 慢慢计算机普及到西欧其他地区, 发现好多字符没法识别处理; 于是对ASCII进行了扩展, 叫EASCII编码; 还是一个字节, 从128 – 255; 但是针对于这一块的扩充, 各个厂家各有自己的标准(比如当时比较有名的CP437); 最后导致互相之间没法沟通;
  所以, 后来, 由国际标准化组织(ISO) 以及国际电工委员会(IEC)联合制定了一个标准ISO/8859-1(Latin-1), 继承了CP437的128-159; 重新定义了160-255;

- 3. 然后到中国之后, 全都懵逼了; 汉字博大精深, 一个字节肯定不够; so, 国人自己搞了一个GB2312来存储中文, 6763个汉字;(双字节, 兼容ASCII)
  可是, 一开始还很爽; 后来发现还有繁体字, 藏文, 蒙文, 维吾尔文... 懵逼X2; 于是一狠心, 搞了一个GBK, 全给他们搞进来;

- 4. 中国是搞定了, 那日本,韩国... ... 如果到时候, 各有各的字符编码, 那该怎样沟通?比如 666 , 在中国代表NB; 在岛国代表SB, 那就乱套了;
  所以, 统一联盟国际组织, 提出了Unicode编码; 涵盖了世界上所有的文字, 每一个字符都有对应的唯一一个字符码, 这回大家都开心了
  但是, 针对于每个字符码, 使用几个字节存储的问题, 又存在几个不同的具体解决方案; 比如utf-8, utf-16, utf-32... 所以, 其实, 我们讨论这边编码的时候, 都是指Unicode编码

`utf-16` 通通使用两个字节存储
`utf-8` 有的使用一个字节，有的使用两个字节存储，甚至三个

解决乱码的方法

- 保证编码与解码都使用同一个字符编码
- 编码格式要支持要使用的字符（比如 `ASCII` 就不支持中文）

完成于 `201810261228`