**1.**

**包 package：** 为避免模块名冲突，Python引入了按目录组织模块的方法，称之为 包（package）。包是含有Python模块的文件夹。]

当一个文件夹下有init.py时，意为该文件夹是一个包（package），其下的多个模块（module）构成一个整体，而这些模块（module）都可通过同一个包（package）导入其他代码中.

2.

sys.modules 是一个 将模块名称（module\_name）映射到已加载的模块（modules） 的字典。可用来强制重新加载modules。Python一启动，它将被加载在内存中。

当我们导入新modules，sys.modules将自动记录下该module；当第二次再导入该module时，Python将直接到字典中查找，加快运行速度。

它是个字典，故拥有字典的一切方法，如sys.modules.keys()、sys.modules.values()、sys.modules[‘os’]。但请不要轻易替换字典、或从字典中删除某元素，将可能导致Python运行失败。

import sys

print(sys.modules)#打印，查看该字典具体内容。

3.

Python在寻找需要加载的模块路径时候，在sys.path所指示的路径搜索。如果我们要改变拓展需要找完的模块路径，我们可以直接对sys.path进行操作，进行sys.path.append(“..”), 也可以在当前运行环境中对系统变量PYTHONPATH进行改变。(注意对PYTHONPATH系统变量的改变，如果使用export进行改变将会是一次性的，除非放入~/.bashrc当中，这和Linux环境设置相关)

4.

当引入模块的时候，模块中的方法将会运行。除非我们在模块中写入if \_\_name\_\_ == “main”, 否则默认将会执行一遍。

5.

默认的sys.path, Python搜索的模块是包含本目录的，但是并不包含上一级的目录。

6.

\_\_all\_\_变量。

如果这个\_\_all\_\_变量设置在包级别，也就是设置在\_\_init\_\_.py中，那么from 这个包 import \*, 只会引入\_\_all\_\_中定义的模块，如果\_\_all\_\_不指定模块，那么不引入任何模块。

如果这个\_\_all\_\_变量设置在模块级别，也就是在xx.py中，那么如果\_\_all\_\_不指定引入具体方法或者变量，则from 模块名 import \*将引入所有。如果指定，则引入指定的变量或者方法。

7.

python 中只能在package中使用相对导入，不能在用户的应用程序中使用相对导入，因为不论是相对导入还是绝对导入，都是相当于当前模块来说的，对于用户的主应用程序，也就是入口文件，模块名总是为“ \_\_main\_\_  ”, 所以用户的应用程序必须使用绝对导入，而package中的导入可以使用相对导入。

8.

├── mdl

│ ├── \_\_init\_\_.py

│ ├── bar.py

│ └── foo.py

└── test.py

在Python3中，加入在mdl文件夹中的\_\_init\_\_.py文件，要想引入bar.py中的Bar方法，使用from .bar import Bar，这样Pyhon2和Python3都可以接受。 如果采用from bar import Bar这样Python3无法接受，只有Python2接受。

9.

Python中相对导入，绝对导入的不是import,或者from ..来区别。如果在引入中使用了., ..，表示上层package则就是相对导入，其他为绝对导入。