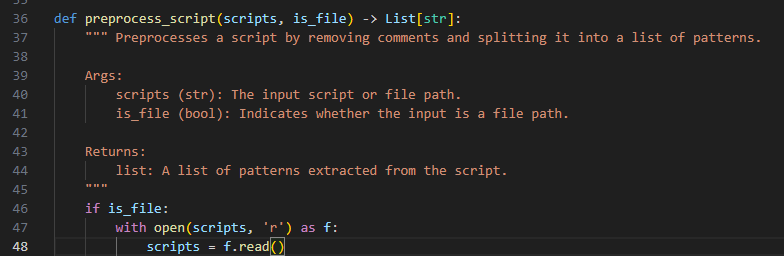
# 命令预处理

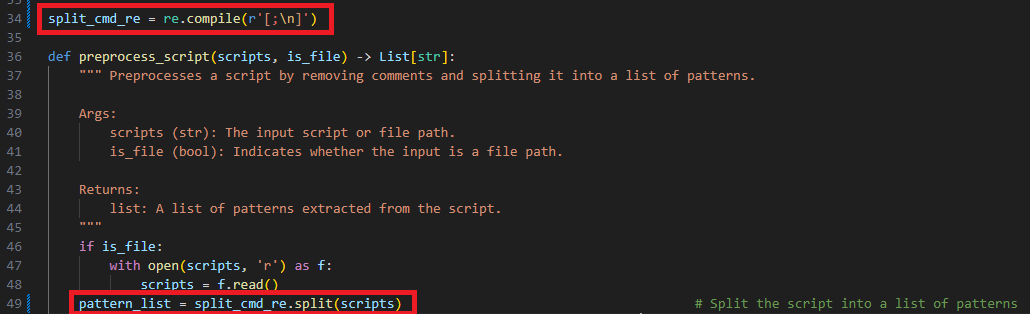
## 命令传入

sed-command 可以通过命令行参数或读取文件获得。如果传入了文件路径，则读取。



## 命令分割

传入的命令可能是单个，也可能是多个。如果是多个命令，是以换行或分号来间隔的，分割也就是换行符和分号进行。

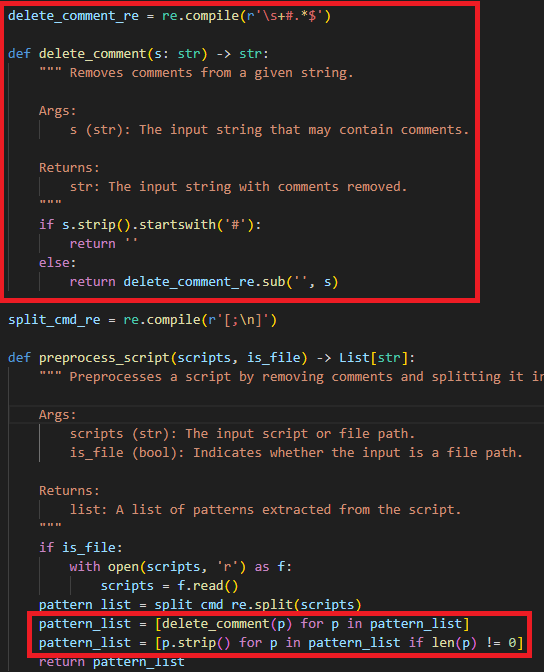


## 去除多余空格和注释

上一步完成了命令分割，现在对分割好的单个命令去除空格和注释处理。

如果存在一行只有注释，这种情况判断是不是“#”开头，是就直接返回空字符串。如果不是，那就存在命令和注释混合，使用正则表达式替换去除注释。

在完成注释处理以后，移除空字符串（仅含注释的），并去除命令字段首尾的空格。

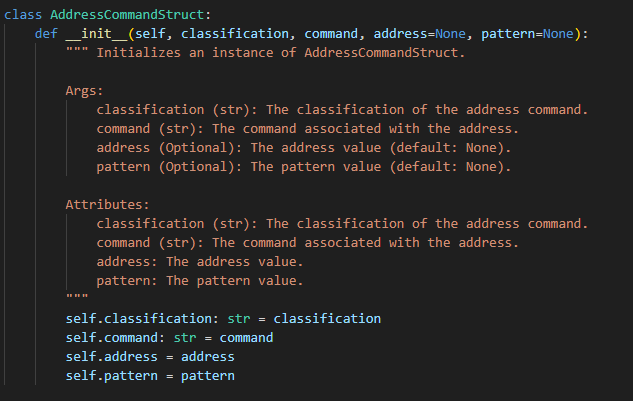


# 2 命令字段分割

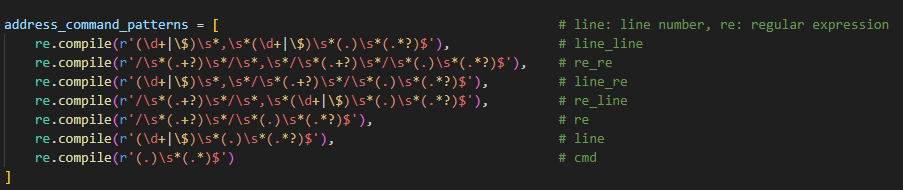
实现的命令有q、p、s、d。命令前方可能存在地址，也可能没有，共有7种情况。

s命令后存在替换规则字段，其它三个命令没有。

于是定义了一个结构来存储每个命令，classification 是命令类别，根据地址类型来分类的；command 用于存储纯命令，即q、p、s和d中的一种；address 用于存储地址；pattern用于存储s命令的替换规则。

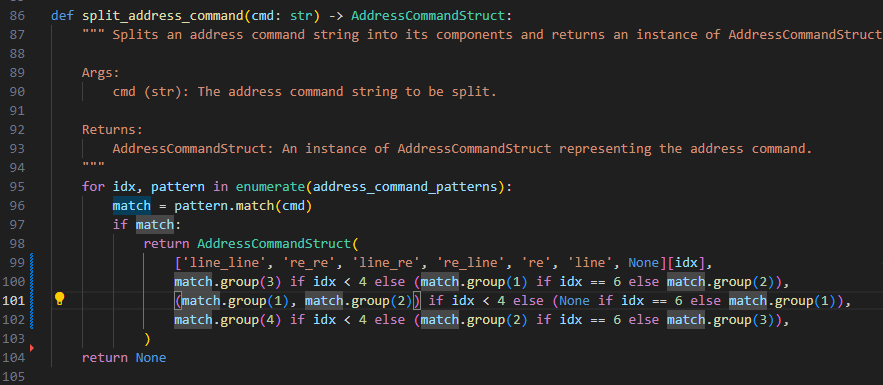


地址部分，可能无地址，指定行数 、正则表达式匹配、范围（行数范围、正则表达式范围以及行数和正则表达式混合的范围）。classification具体分为为：无地址（None）、指定行数地址（line）、行数范围地址（line\_line）、正则表达式（re）、正则表达式范围（re\_re）、行数起始-正则表达式结束范围（line\_re）、正则表达式起始-行数结束范围（re\_line）。分别使用以下正则表达式匹配：

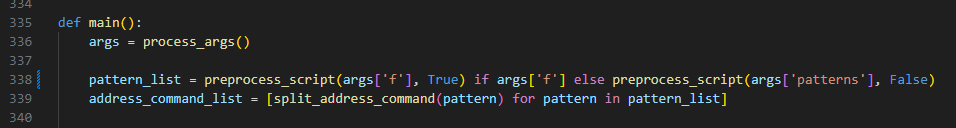


如果地址是范围的，会把起始和结束分别提取出来作为一个元组保存，如(start,end)。其中s命令后部有替换规则，上面正则表达式匹配的时候会把它单独提取出来。

下面的这个函数就是遍历上面的正则表达式进行匹配分割，并把分割结果保存到上面定义的 AddressCommandStruct 结构中。



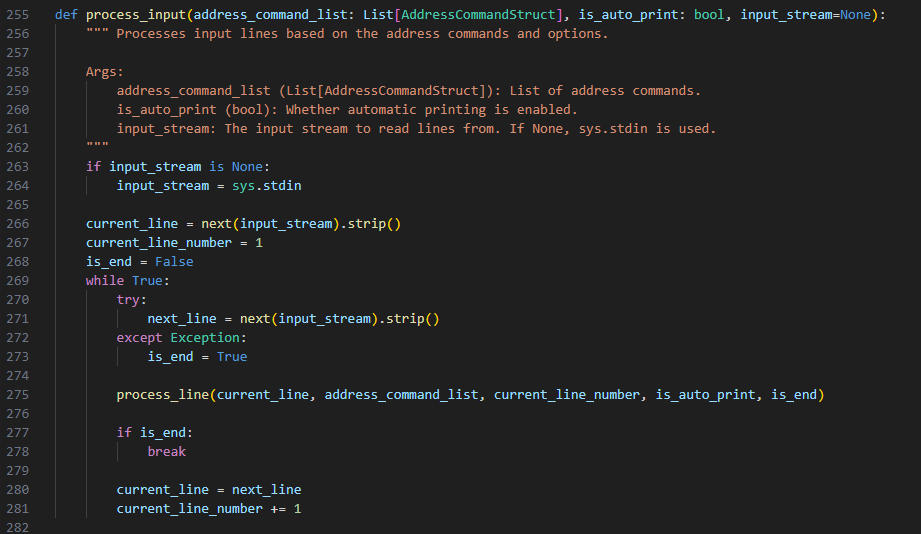
main 函数这里就是将参数传入命令预处理，获得去除注释和空格的单个命令字符串，再进一步调用命令分割，获得一个包含AddressCommandStruct结构的列表。



# 3 命令执行

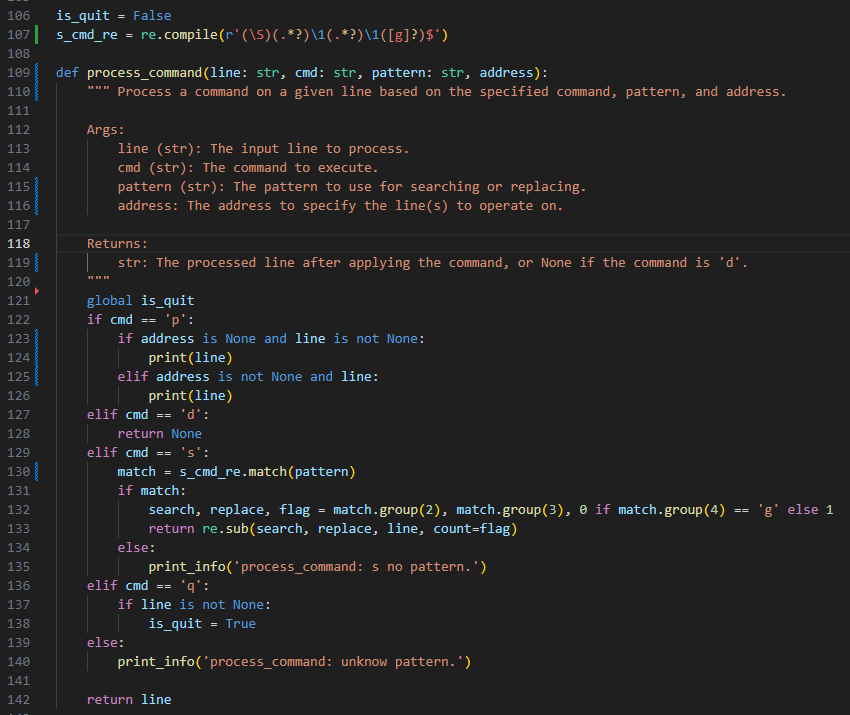
## 3.1 待处理内容读取

如果指定了文件路径，就打开文件，如果没有指定文件路径就从标准输入stdin读取。先预读取一行，然后循环读取下一行，读取下一行失败则代表已经成功读取的一行是最后一行。这个是否为最后一行的标志用于命令中“$”末行地址匹配，同时用于判断是否停止读取。



## 3.2 命令处理

实现 p 命令打印操作，如果地址为空，则即使当前处理行为空字符串也要打印，地址不为空，则不打印空字符串；s命令替换操作，先使用正则表达式分割出匹配内容、替换内容和全局替换标志g，再用 Python 正则表达式替换执行替换操作；q设置退出标志，d 返回 None 标志当前行删除。



## 3.3 地址匹配

地址匹配在 process\_line 函数中执行，前面每读取一行待处理内容就会调用process\_line 执行，process\_line 根据地址匹配决定当前行是否调用 process\_command 执行对应命令。

“-n”选项的判断也在这里，如果使用了“-n”参数则不打印该行。

另外如果执行了d命令返回None，则删除了当前行的内容；如果执行了q设置了退出标志，会在process\_line 中执行 exit(0) 退出。