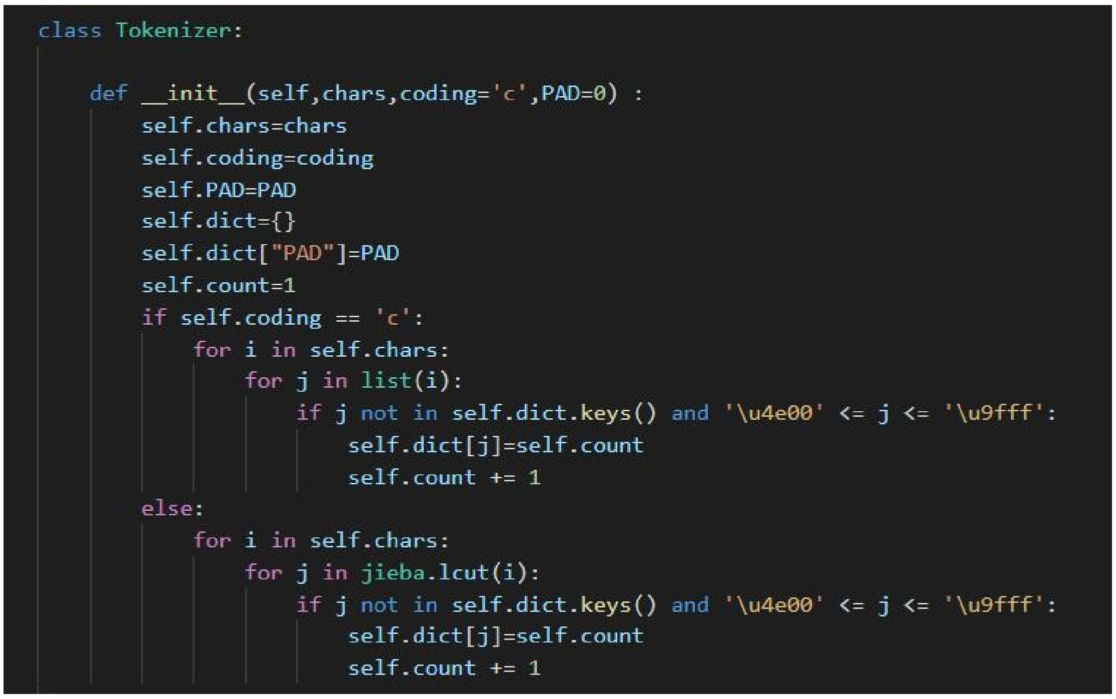
1. 进行初始化

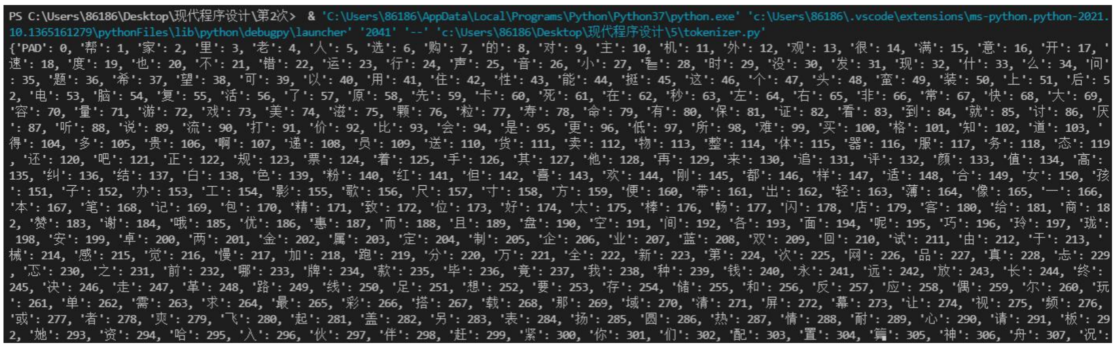
传递进来字符串列表后，此时根据模式（w 为词模式，c 为字模式）判断进行词典的初始化。同时要判断字符是否为汉字，如果不是汉字不记录进入词典



```python
class Tokenizer:

    def __init__(self,chars,coding='c',PAD=0) :
        self.chars=chars
        self.coding=coding
        self.PAD=PAD
        self.dict={}
        self.dict["PAD"]=PAD
        self.count=1
        if self.coding == 'c':
            for i in self.chars:
                for j in list(i):
                    if j not in self.dict.keys() and '\u4e00' <= j <= '\u9fff':
                        self.dict[j]=self.count
                        self.count += 1
        else:
            for i in self.chars:
                for j in jieba.lcut(i):
                    if j not in self.dict.keys() and '\u4e00' <= j <= '\u9fff':
                        self.dict[j]=self.count
                        self.count += 1
```

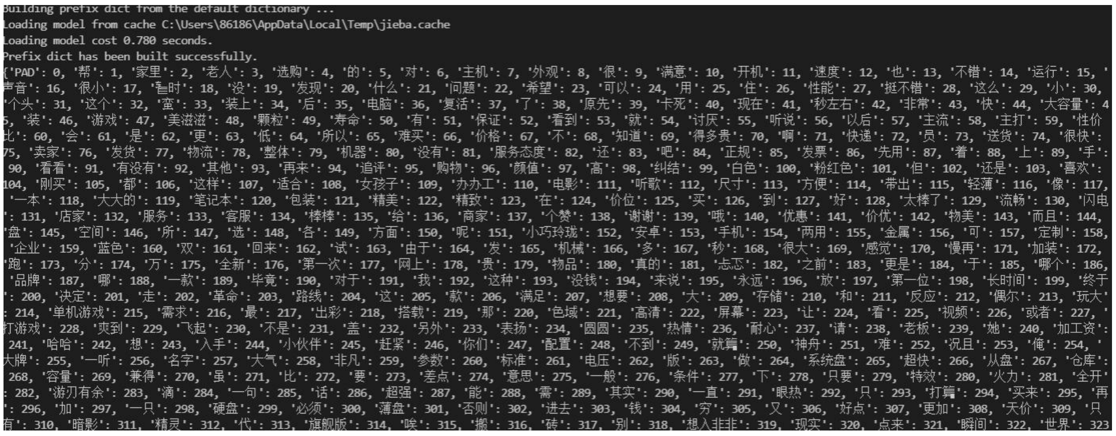
运行结果:

字模式:



```
PS C:\Users\86186\Desktop\现代程序设计\第2次>  & 'C:\Users\86186\AppData\Local\Programs\Python\Python37\python.exe' 'c:\Users\86186\.vscode\extensions\ms-python.python-2021.
10.1365161279\pythonFiles\lib\python\debugpy\launcher' '2041' '--' 'c:\Users\86186\Desktop\现代程序设计\5\tokenizer.py'
{'PAD': 0, '帮': 1, '家': 2, '里': 3, '老': 4, '人': 5, '选': 6, '购': 7, '的': 8, '对': 9, '主': 10, '机': 11, '外': 12, '观': 13, '很': 14, '满': 15, '意': 16, '开': 17, '
速': 18, '度': 19, '也': 20, '不': 21, '错': 22, '运': 23, '行': 24, '声': 25, '音': 26, '小': 27, '耄': 28, '时': 29, '没': 30, '发': 31, '现': 32, '什': 33, '么': 34, '问'
: 35, '题': 36, '希': 37, '望': 38, '可': 39, '以': 40, '用': 41, '住': 42, '性': 43, '能': 44, '挺': 45, '这': 46, '个': 47, '头': 48, '蛮': 49, '装': 50, '上': 51, '后': 5
2, '电': 53, '脑': 54, '复': 55, '活': 56, '了': 57, '原': 58, '先': 59, '卡': 60, '死': 61, '在': 62, '秒': 63, '左': 64, '右': 65, '非': 66, '常': 67, '快': 68, '大': 69,
'容': 70, '量': 71, '游': 72, '戏': 73, '美': 74, '滋': 75, '颗': 76, '粒': 77, '寿': 78, '命': 79, '有': 80, '保': 81, '证': 82, '看': 83, '到': 84, '就': 85, '讨': 86, '厌
': 87, '听': 88, '说': 89, '流': 90, '打': 91, '价': 92, '比': 93, '会': 94, '是': 95, '更': 96, '低': 97, '所': 98, '难': 99, '买': 100, '格': 101, '知': 102, '道': 103, '
得': 104, '多': 105, '贵': 106, '啊': 107, '递': 108, '员': 109, '送': 110, '货': 111, '卖': 112, '物': 113, '整': 114, '体': 115, '器': 116, '服': 117, '务': 118, '态': 119
, '还': 120, '吧': 121, '正': 122, '规': 123, '票': 124, '着': 125, '手': 126, '其': 127, '他': 128, '再': 129, '来': 130, '追': 131, '评': 132, '颜': 133, '值': 134, '高':
135, '纠': 136, '结': 137, '白': 138, '色': 139, '粉': 140, '红': 141, '但': 142, '喜': 143, '欢': 144, '刚': 145, '都': 146, '样': 147, '适': 148, '合': 149, '女': 150, '孩
': 151, '子': 152, '办': 153, '工': 154, '影': 155, '歌': 156, '尺': 157, '寸': 158, '方': 159, '便': 160, '带': 161, '出': 162, '轻': 163, '薄': 164, '像': 165, '一': 166,
'本': 167, '笔': 168, '记': 169, '包': 170, '精': 171, '致': 172, '位': 173, '好': 174, '太': 175, '棒': 176, '畅': 177, '闪': 178, '店': 179, '客': 180, '给': 181, '商': 18
2, '赞': 183, '谢': 184, '哦': 185, '优': 186, '惠': 187, '而': 188, '且': 189, '盘': 190, '空': 191, '间': 192, '各': 193, '面': 194, '呢': 195, '巧': 196, '玲': 197, '珑':
 198, '安': 199, '卓': 200, '两': 201, '金': 202, '属': 203, '定': 204, '制': 205, '企': 206, '业': 207, '蓝': 208, '双': 209, '回': 210, '试': 211, '由': 212, '于': 213, '
械': 214, '感': 215, '觉': 216, '慢': 217, '加': 218, '跑': 219, '分': 220, '万': 221, '全': 222, '新': 223, '第': 224, '次': 225, '网': 226, '品': 227, '真': 228, '忐': 229
, '忑': 230, '之': 231, '前': 232, '哪': 233, '牌': 234, '款': 235, '毕': 236, '竟': 237, '我': 238, '种': 239, '钱': 240, '永': 241, '远': 242, '放': 243, '长': 244, '终':
245, '决': 246, '走': 247, '革': 248, '路': 249, '线': 250, '足': 251, '想': 252, '要': 253, '存': 254, '储': 255, '和': 256, '反': 257, '应': 258, '偶': 259, '尔': 260, '玩
': 261, '单': 262, '需': 263, '求': 264, '最': 265, '彩': 266, '搭': 267, '载': 268, '那': 269, '域': 270, '清': 271, '屏': 272, '幕': 273, '让': 274, '视': 275, '频': 276,
'或': 277, '者': 278, '爽': 279, '飞': 280, '起': 281, '盖': 282, '另': 283, '表': 284, '扬': 285, '圆': 286, '热': 287, '情': 288, '耐': 289, '心': 290, '请': 291, '板': 29
2, '她': 293, '资': 294, '哈': 295, '入': 296, '伙': 297, '伴': 298, '赶': 299, '紧': 300, '你': 301, '们': 302, '配': 303, '置': 304, '算': 305, '神': 306, '舟': 307, '况':
```

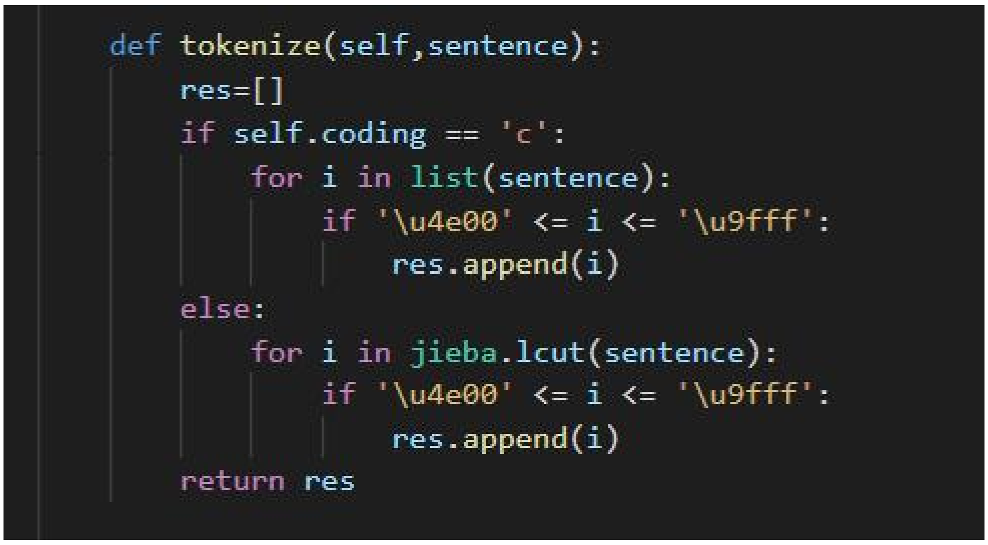
词模式:



```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\86186\AppData\Local\Temp\jieba.cache
Loading model cost 0.780 seconds.
Prefix dict has been built successfully.
{'PAD': 0, '帮': 1, '家里': 2, '老人': 3, '选购': 4, '的': 5, '对': 6, '主机': 7, '外观': 8, '很': 9, '满意': 10, '开机': 11, '速度': 12, '也': 13, '不错': 14, '运行': 15, '
声音': 16, '很小': 17, '耄时': 18, '没': 19, '发现': 20, '什么': 21, '问题': 22, '希望': 23, '可以': 24, '用': 25, '住': 26, '性能': 27, '挺不错': 28, '这么': 29, '小': 30,
'个头': 31, '这个': 32, '蛮': 33, '装上': 34, '后': 35, '电脑': 36, '复活': 37, '了': 38, '原先': 39, '卡死': 40, '现在': 41, '秒左右': 42, '非常': 43, '快': 44, '大容量': 4
5, '装': 46, '游戏': 47, '美滋滋': 48, '颗粒': 49, '寿命': 50, '有': 51, '保证': 52, '看到': 53, '就': 54, '讨厌': 55, '听说': 56, '以后': 57, '主流': 58, '主打': 59, '性价
比': 60, '会': 61, '是': 62, '更': 63, '低': 64, '所以': 65, '难买': 66, '价格': 67, '不': 68, '知道': 69, '得多贵': 70, '啊': 71, '快递': 72, '员': 73, '送货': 74, '很快':
75, '卖家': 76, '发货': 77, '物流': 78, '整体': 79, '机器': 80, '没有': 81, '服务态度': 82, '还': 83, '吧': 84, '正规': 85, '发票': 86, '先用': 87, '着': 88, '上': 89, '手':
 90, '看看': 91, '有没有': 92, '其他': 93, '再来': 94, '追评': 95, '购物': 96, '颜值': 97, '高': 98, '纠结': 99, '白色': 100, '粉红色': 101, '但': 102, '还是': 103, '喜欢':
104, '刚买': 105, '都': 106, '这样': 107, '适合': 108, '女孩子': 109, '办办工': 110, '电影': 111, '听歌': 112, '尺寸': 113, '方便': 114, '带出': 115, '轻薄': 116, '像': 117,
 '一本': 118, '大大的': 119, '笔记本': 120, '包装': 121, '精美': 122, '精致': 123, '在': 124, '价位': 125, '买': 126, '到': 127, '好': 128, '太棒了': 129, '流畅': 130, '闪电
': 131, '店家': 132, '服务': 133, '客服': 134, '棒棒': 135, '给': 136, '商家': 137, '个赞': 138, '谢谢': 139, '哦': 140, '优惠': 141, '价优': 142, '物美': 143, '而且': 144,
'盘': 145, '空间': 146, '所': 147, '选': 148, '各': 149, '方面': 150, '呢': 151, '小巧玲珑': 152, '安卓': 153, '手机': 154, '两用': 155, '金属': 156, '可': 157, '定制': 158,
 '企业': 159, '蓝色': 160, '双': 161, '回来': 162, '试': 163, '由于': 164, '发': 165, '机械': 166, '多': 167, '秒': 168, '很大': 169, '感觉': 170, '慢再': 171, '加装': 172,
'跑': 173, '分': 174, '万': 175, '全新': 176, '第一次': 177, '网上': 178, '贵': 179, '物品': 180, '真的': 181, '忐忑': 182, '之前': 183, '更是': 184, '于': 185, '哪个': 186,
 '品牌': 187, '哪': 188, '一款': 189, '毕竟': 190, '对于': 191, '我': 192, '这种': 193, '没钱': 194, '来说': 195, '永远': 196, '放': 197, '第一位': 198, '长时间': 199, '终于
': 200, '决定': 201, '走': 202, '革命': 203, '路线': 204, '这': 205, '款': 206, '满足': 207, '想要': 208, '大': 209, '存储': 210, '和': 211, '反应': 212, '偶尔': 213, '玩大'
: 214, '单机游戏': 215, '需求': 216, '最': 217, '出彩': 218, '搭载': 219, '那': 220, '色域': 221, '高清': 222, '屏幕': 223, '让': 224, '看': 225, '视频': 226, '或者': 227, '
打游戏': 228, '爽到': 229, '飞起': 230, '不是': 231, '盖': 232, '另外': 233, '表扬': 234, '圆圆': 235, '热情': 236, '耐心': 237, '请': 238, '老板': 239, '她': 240, '加工资':
 241, '哈哈': 242, '想': 243, '入手': 244, '小伙伴': 245, '赶紧': 246, '你们': 247, '配置': 248, '不到': 249, '就算': 250, '神舟': 251, '难': 252, '况且': 253, '俺': 254, '
大牌': 255, '一听': 256, '名字': 257, '大气': 258, '非凡': 259, '参数': 260, '标准': 261, '电压': 262, '版': 263, '做': 264, '系统盘': 265, '超快': 266, '从盘': 267, '仓库':
 268, '容量': 269, '兼得': 270, '虽': 271, '比': 272, '要': 273, '差点': 274, '意思': 275, '一般': 276, '条件': 277, '下': 278, '只要': 279, '特效': 280, '火力': 281, '全开'
: 282, '游刃有余': 283, '滴': 284, '一句': 285, '话': 286, '超强': 287, '能': 288, '需': 289, '其实': 290, '一直': 291, '眼热': 292, '只': 293, '打算': 294, '买来': 295, '再
': 296, '加': 297, '一只': 298, '硬盘': 299, '必须': 300, '薄盘': 301, '否则': 302, '进去': 303, '钱': 304, '穷': 305, '又': 306, '好点': 307, '更加': 308, '天价': 309, '只
有': 310, '暗影': 311, '精灵': 312, '代': 313, '旗舰版': 314, '唉': 315, '搬': 316, '砖': 317, '别': 318, '想入非非': 319, '现实': 320, '点来': 321, '瞬间': 322, '世界': 323
```

2. 2. tokenize(self, sentence) 输入一句话，返回分词(字）后的字符列表(list_of_chars)。

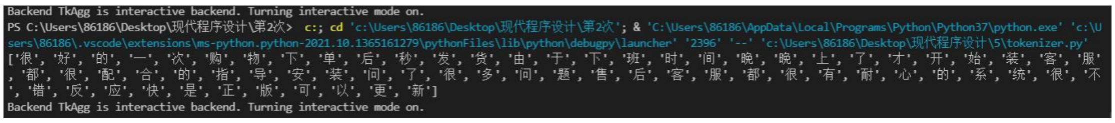
根据不同的模式进行分词（分字），同时筛除不是汉字的字符



```python
def tokenize(self,sentence):
    res=[]
    if self.coding == 'c':
        for i in list(sentence):
            if '\u4e00' <= i <= '\u9fff':
                res.append(i)
    else:
        for i in jieba.lcut(sentence):
            if '\u4e00' <= i <= '\u9fff':
                res.append(i)
    return res
```

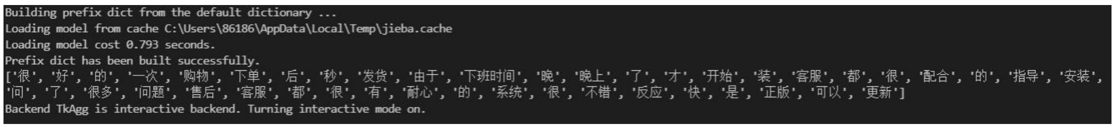
运行结果:

字模式:



```
Backend TkAgg is interactive backend. Turning interactive mode on.
PS C:\Users\86186\Desktop\现代程序设计\第2次>  c:; cd 'c:\Users\86186\Desktop\现代程序设计\第2次'; & 'C:\Users\86186\AppData\Local\Programs\Python\Python37\python.exe' 'c:\U
sers\86186\.vscode\extensions\ms-python.python-2021.10.1365161279\pythonFiles\lib\python\debugpy\launcher' '2396' '--' 'c:\Users\86186\Desktop\现代程序设计\5\tokenizer.py'
['很', '好', '的', '一', '次', '购', '物', '下', '单', '后', '秒', '发', '货', '由', '于', '下', '班', '时', '间', '晚', '晚', '上', '了', '才', '开', '始', '装', '客', '服'
, '都', '很', '配', '合', '的', '指', '导', '安', '装', '问', '了', '很', '多', '问', '题', '售', '后', '客', '服', '都', '很', '有', '耐', '心', '的', '系', '统', '很', '不
', '错', '反', '应', '快', '是', '正', '版', '可', '以', '更', '新']
Backend TkAgg is interactive backend. Turning interactive mode on.
```

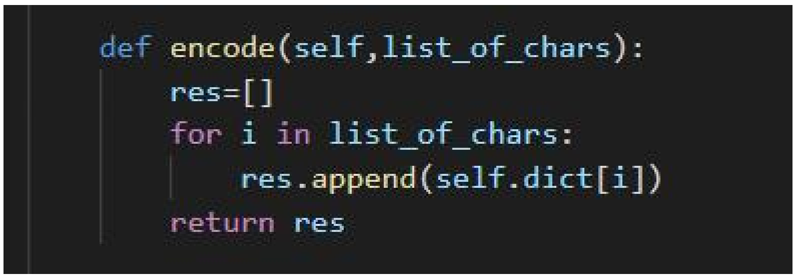
词模式:



```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\86186\AppData\Local\Temp\jieba.cache
Loading model cost 0.793 seconds.
Prefix dict has been built successfully.
['很', '好', '的', '一次', '购物', '下单', '后', '秒', '发货', '由于', '下班时间', '晚', '晚上', '了', '才', '开始', '装', '客服', '都', '很', '配合', '的', '指导', '安装',
'问', '了', '很多', '问题', '售后', '客服', '都', '很', '有', '耐心', '的', '系统', '很', '不错', '反应', '快', '是', '正版', '可以', '更新']
Backend TkAgg is interactive backend. Turning interactive mode on.
```

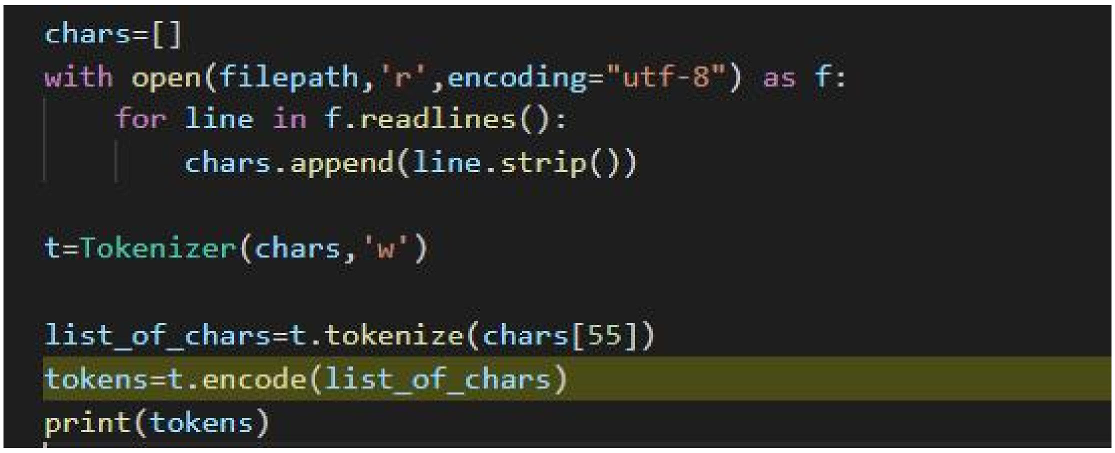
3. encode(self, list_of_chars) 输入字符(字或者词）的字符列表，返回转换后的数字列表(tokens)

根据词典生成相应的数字列表

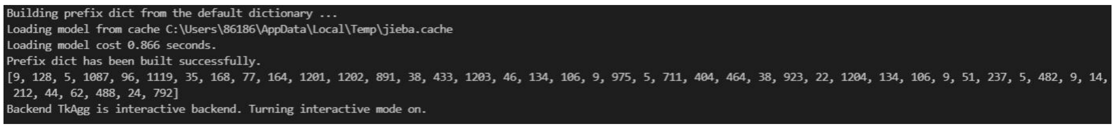


```python
def encode(self,list_of_chars):
    res=[]
    for i in list_of_chars:
        res.append(self.dict[i])
    return res
```

运行结果:

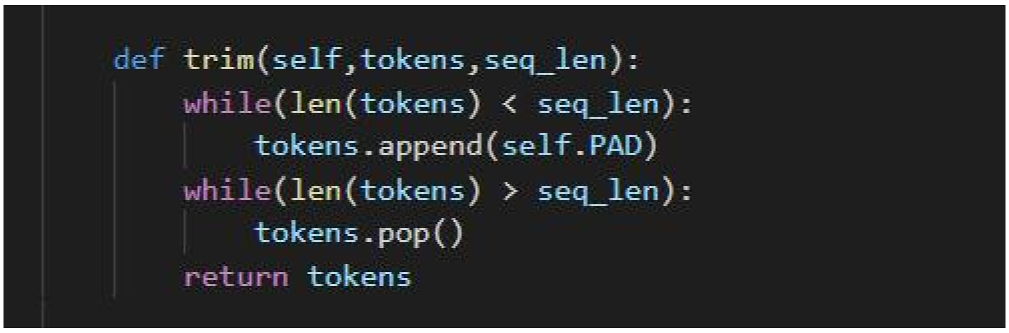


```
chars=[]
with open(filepath,'r',encoding="utf-8") as f:
    for line in f.readlines():
        chars.append(line.strip())

t=Tokenizer(chars,'w')

list_of_chars=t.tokenize(chars[55])
tokens=t.encode(list_of_chars)
print(tokens)
```



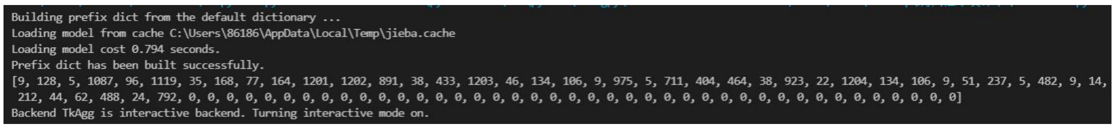
```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\86186\AppData\Local\Temp\jieba.cache
Loading model cost 0.866 seconds.
Prefix dict has been built successfully.
[9, 128, 5, 1087, 96, 1119, 35, 168, 77, 164, 1201, 1202, 891, 38, 433, 1203, 46, 134, 106, 9, 975, 5, 711, 404, 464, 38, 923, 22, 1204, 134, 106, 9, 51, 237, 5, 482, 9, 14,
 212, 44, 62, 488, 24, 792]
Backend TkAgg is interactive backend. Turning interactive mode on.
```

4. trim(self, tokens, seq_len) 输入数字列表 tokens，整理数字列表的长度。不足 seq_len 的部分用 PAD 补足，超过的部分截断。



```
def trim(self,tokens,seq_len):
    while(len(tokens) < seq_len):
        tokens.append(self.PAD)
    while(len(tokens) > seq_len):
        tokens.pop()
    return tokens
```

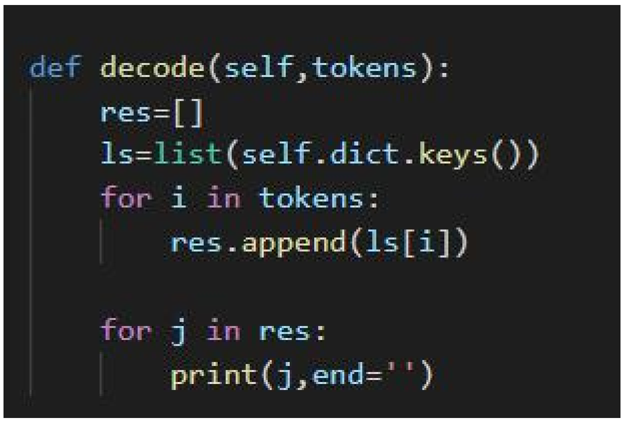
运行结果:



```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\86186\AppData\Local\Temp\jieba.cache
Loading model cost 0.794 seconds.
Prefix dict has been built successfully.
[9, 128, 5, 1087, 96, 1119, 35, 168, 77, 164, 1201, 1202, 891, 38, 433, 1203, 46, 134, 106, 9, 975, 5, 711, 404, 464, 38, 923, 22, 1204, 134, 106, 9, 51, 237, 5, 482, 9, 14,
 212, 44, 62, 488, 24, 792, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Backend TkAgg is interactive backend. Turning interactive mode on.
```

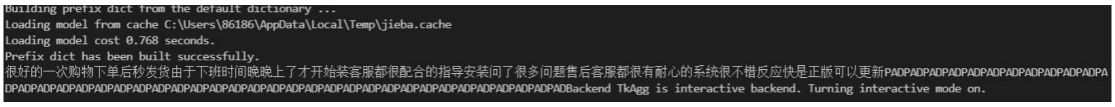
5. decode(self, tokens) 将模型输出的数字列表翻译回句子。如果有 PAD，输出'[PAD]'。

注意到词典中的值刚好对应键的索引值，故生成键的列表，利用键值当作索引值



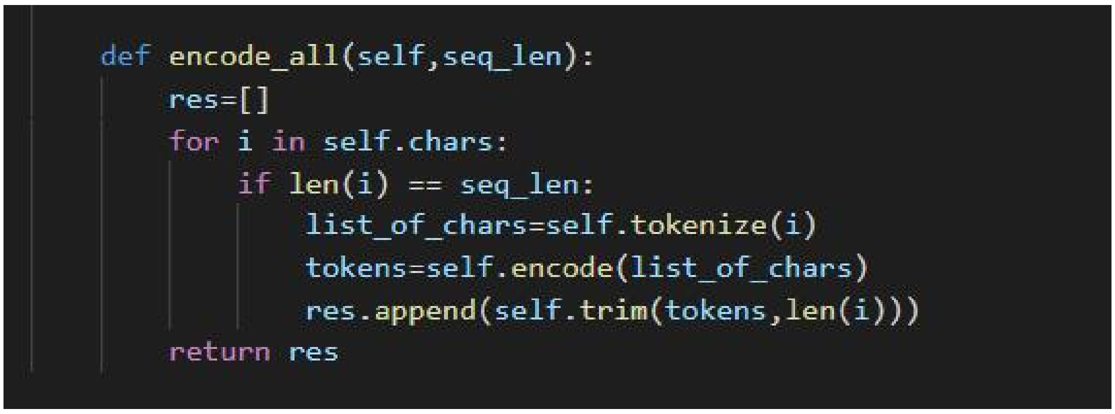
```
def decode(self,tokens):
    res=[]
    ls=list(self.dict.keys())
    for i in tokens:
        res.append(ls[i])

    for j in res:
        print(j,end='')
```

运行结果:



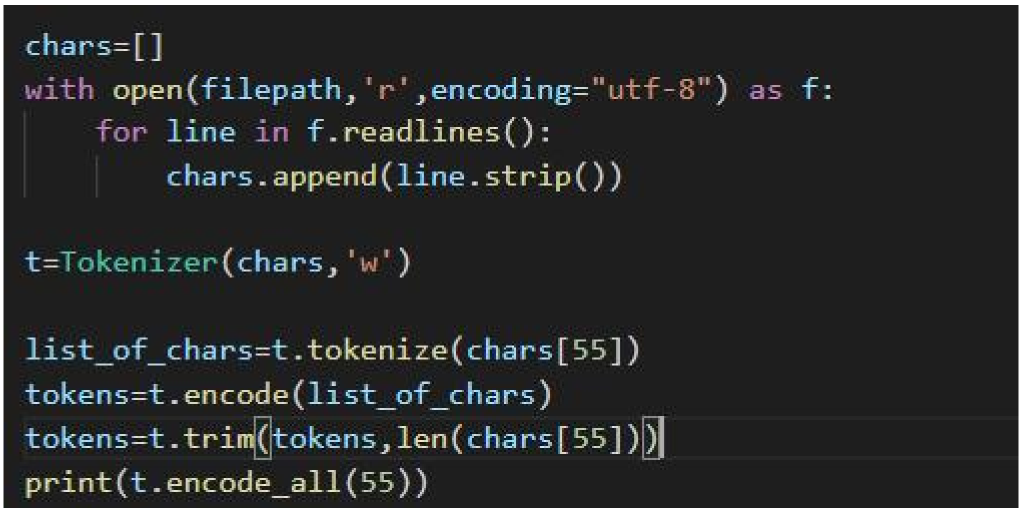
```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\86186\AppData\Local\Temp\jieba.cache
Loading model cost 0.768 seconds.
Prefix dict has been built successfully.
很好的一次购物下单后秒发货由于下班时间晚晚上了才开始装客服都很配合的指导安装问了很多问题售后客服都很有耐心的系统很不错反应快是正版可以更新PADPADPADPADPADPADPADPADPADPADPADPA
DPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADBackend TkAgg is interactive backend. Turning interactive mode on.
```

6. encode_all(self, seq_len) 返回所有文本 (chars)的长度为 seq_len 的 tokens。

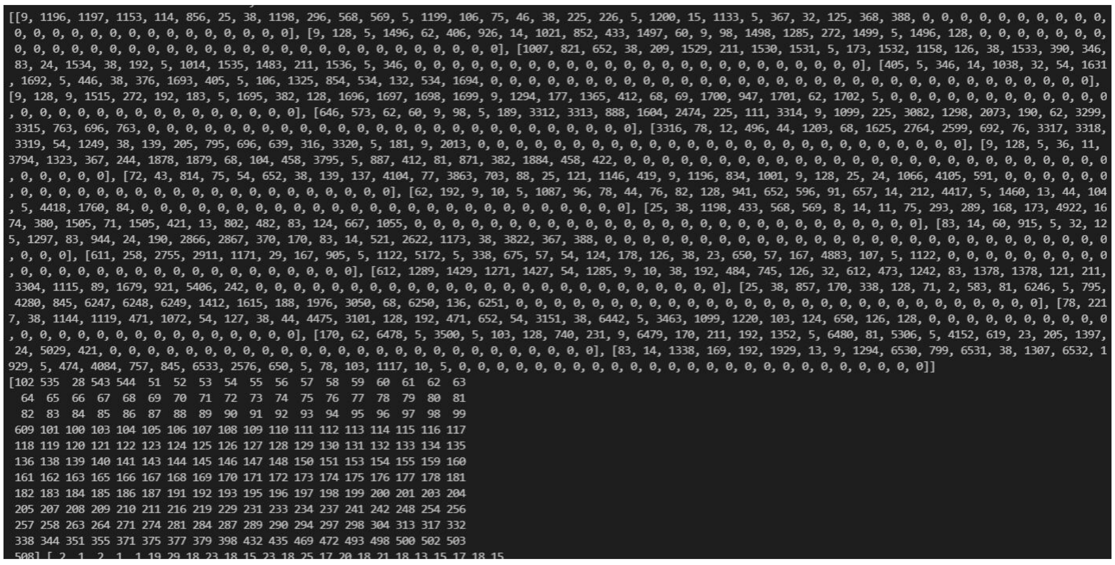


```python
    def encode_all(self,seq_len):
        res=[]
        for i in self.chars:
            if len(i) == seq_len:
                list_of_chars=self.tokenize(i)
                tokens=self.encode(list_of_chars)
                res.append(self.trim(tokens,len(i)))
        return res
```

运行结果:



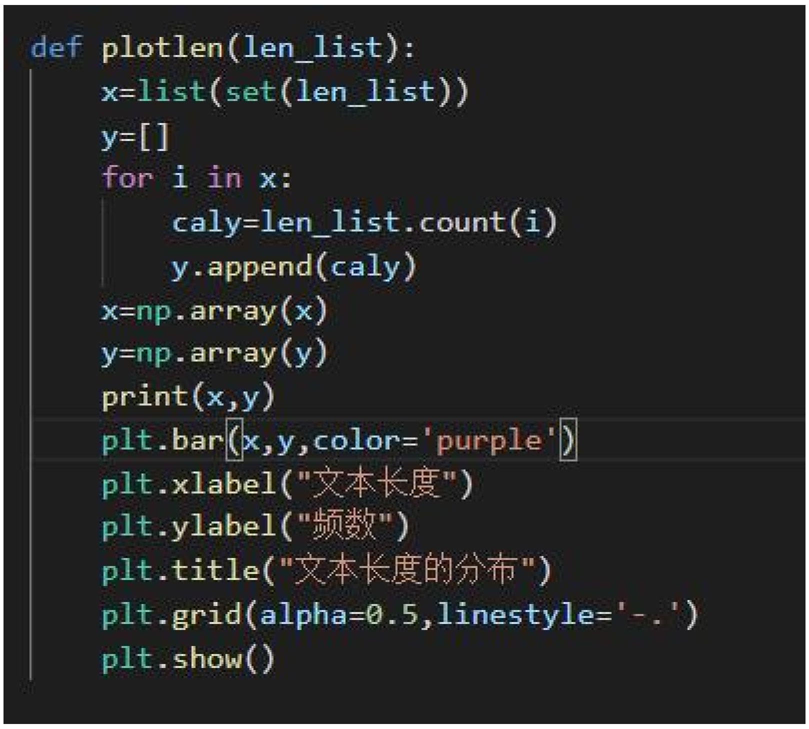
```python
chars=[]
with open(filepath,'r',encoding="utf-8") as f:
    for line in f.readlines():
        chars.append(line.strip())

t=Tokenizer(chars,'w')

list_of_chars=t.tokenize(chars[55])
tokens=t.encode(list_of_chars)
tokens=t.trim(tokens,len(chars[55]))
print(t.encode_all(55))
```



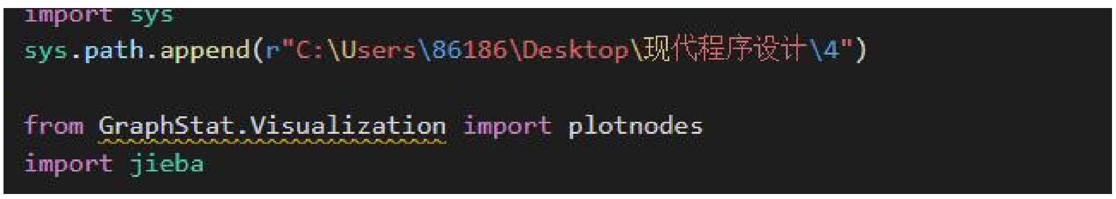
```
[[9, 1196, 1197, 1153, 114, 856, 25, 38, 1198, 296, 568, 569, 5, 1199, 106, 75, 46, 38, 225, 226, 5, 1200, 15, 1133, 5, 367, 32, 125, 368, 388, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [9, 128, 5, 1496, 62, 406, 926, 14, 1021, 852, 433, 1497, 60, 9, 98, 1498, 1285, 272, 1499, 5, 1496, 128, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1007, 821, 652, 38, 209, 1529, 211, 1530, 1531, 5, 173, 1532, 1158, 126, 38, 1533, 390, 346,
 83, 24, 1534, 38, 192, 5, 1014, 1535, 1483, 211, 1536, 5, 346, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [405, 5, 346, 14, 1038, 32, 54, 1631
, 1692, 5, 446, 38, 376, 1693, 405, 5, 106, 1325, 854, 534, 132, 534, 1694, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[9, 128, 9, 1515, 272, 192, 183, 5, 1695, 382, 128, 1696, 1697, 1698, 1699, 9, 1294, 177, 1365, 412, 68, 69, 1700, 947, 1701, 62, 1702, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [646, 573, 62, 60, 9, 98, 5, 189, 3312, 3313, 888, 1604, 2474, 225, 111, 3314, 9, 1099, 225, 3082, 1298, 2073, 190, 62, 3299,
 3315, 763, 696, 763, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [3316, 78, 12, 496, 44, 1203, 68, 1625, 2764, 2599, 692, 76, 3317, 3318,
 3319, 54, 1249, 38, 139, 205, 795, 696, 639, 316, 3320, 5, 181, 9, 2013, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [9, 128, 5, 36, 11,
3794, 1323, 367, 244, 1878, 1879, 68, 104, 458, 3795, 5, 887, 412, 81, 871, 382, 1884, 458, 422, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0], [72, 43, 814, 75, 54, 652, 38, 139, 137, 4104, 77, 3863, 703, 88, 25, 121, 1146, 419, 9, 1196, 834, 1001, 9, 128, 25, 24, 1066, 4105, 591, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [62, 192, 9, 10, 5, 1087, 96, 78, 44, 76, 82, 128, 941, 652, 596, 91, 657, 14, 212, 4417, 5, 1460, 13, 44, 104
, 5, 4418, 1760, 84, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [25, 38, 1198, 433, 568, 569, 8, 14, 11, 75, 293, 289, 168, 173, 4922, 16
74, 380, 1505, 71, 1505, 421, 13, 802, 482, 83, 124, 667, 1055, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [83, 14, 60, 915, 5, 32, 12
5, 1297, 83, 944, 24, 190, 2866, 2867, 370, 170, 83, 14, 521, 2622, 1173, 38, 3822, 367, 388, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0], [611, 258, 2755, 2911, 1171, 29, 167, 905, 5, 1122, 5172, 5, 338, 675, 57, 54, 124, 178, 126, 38, 23, 650, 57, 167, 4883, 107, 5, 1122, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [612, 1289, 1429, 1271, 1427, 54, 1285, 9, 10, 38, 192, 484, 745, 126, 32, 612, 473, 1242, 83, 1378, 1378, 121, 211,
 3304, 1115, 89, 1679, 921, 5406, 242, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [25, 38, 857, 170, 338, 128, 71, 2, 583, 81, 6246, 5, 795,
 4280, 845, 6247, 6248, 6249, 1412, 1615, 188, 1976, 3050, 68, 6250, 136, 6251, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [78, 221
7, 38, 1144, 1119, 471, 1072, 54, 127, 38, 44, 4475, 3101, 128, 192, 471, 652, 54, 3151, 38, 6442, 5, 3463, 1099, 1220, 103, 124, 650, 126, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [170, 62, 6478, 5, 3500, 5, 103, 128, 740, 231, 9, 6479, 170, 211, 192, 1352, 5, 6480, 81, 5306, 5, 4152, 619, 23, 205, 1397,
 24, 5029, 421, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [83, 14, 1338, 169, 192, 1929, 13, 9, 1294, 6530, 799, 6531, 38, 1307, 6532, 1
929, 5, 474, 4084, 757, 845, 6533, 2576, 650, 5, 78, 103, 1117, 10, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
[102 535  28 543 544  51  52  53  54  55  56  57  58  59  60  61  62  63
  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81
  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99
 609 101 100 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135
 136 138 139 140 141 143 144 145 146 147 148 150 151 153 154 155 159 160
 161 162 163 165 166 167 168 169 170 171 172 173 174 175 176 177 178 181
 182 183 184 185 186 187 191 192 193 195 196 197 198 199 200 201 203 204
 205 207 208 209 210 211 216 219 229 231 233 234 237 241 242 248 254 256
 257 258 263 264 271 274 281 284 287 289 290 294 297 298 304 313 317 332
 338 344 351 355 371 375 377 379 398 432 435 469 472 493 498 500 502 503
 508] [ 2  1  2  1  1 19 29 18 23 18 15 23 18 25 17 20 18 21 18 13 15 17 18 15
```

7. seq_len 是句子的长度，实际任务中一般怎么来确定一个合适的长度，请以前次作业中的评论文本或微博文本为例，通过文本的长度分布来进行观察和讨论。

在上次作业的基础上，在 plotnodes 模块中加入对文本长度分布的函数，同时在本次作业的环境变量中加入包的路径，导入对应的模块

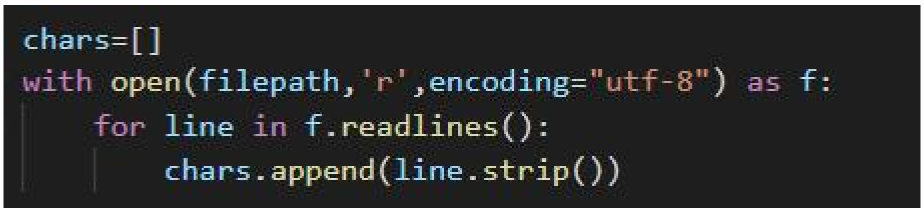


```python
def plotlen(len_list):
    x=list(set(len_list))
    y=[]
    for i in x:
        caly=len_list.count(i)
        y.append(caly)
    x=np.array(x)
    y=np.array(y)
    print(x,y)
    plt.bar(x,y,color='purple')
    plt.xlabel("文本长度")
    plt.ylabel("频数")
    plt.title("文本长度的分布")
    plt.grid(alpha=0.5,linestyle='-.')
    plt.show()
```

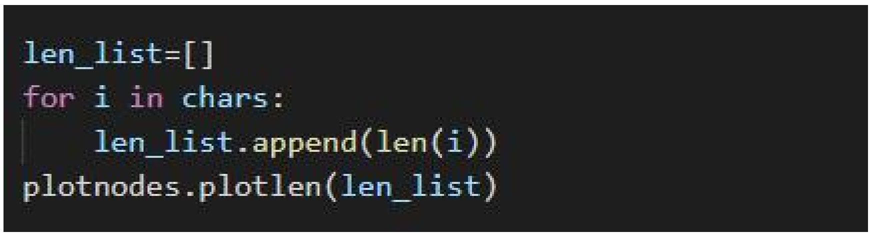


```python
import sys
sys.path.append(r"C:\Users\86186\Desktop\现代程序设计\4")

from GraphStat.Visualization import plotnodes
import jieba
```

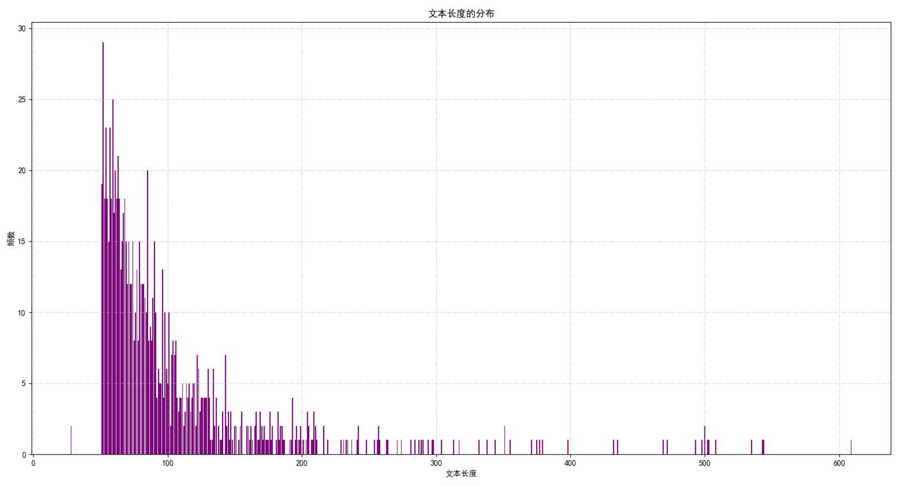
运行结果：



```python
chars=[]
with open(filepath,'r',encoding="utf-8") as f:
    for line in f.readlines():
        chars.append(line.strip())
```



```python
len_list=[]
for i in chars:
    len_list.append(len(i))
plotnodes.plotlen(len_list)
```



将图放大可以发现，京东的评论中，长度为 52 的频数最多，为 29 次