

已修改 2022-08-25 已创建 2022-07-27

1. 格式化代码 【Ctrl + Alt + L】

```
def x():
    a=1
    b=[1,2,3,3,3,3,3,3,3,3,123,12,31,231,23,123,1,231,23,123,1,43,53,643,53,4,24,12,31,231,23,123,24,53,4534,2,1231,23,1]
    print(a)
x()
```

```
def x():
    a = 1
    b = [1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 123, 12, 31, 231, 23, 123, 1, 231, 23,
123, 1, 43, 53, 643, 53, 4, 24, 12, 31,
231, 23, 123, 24, 53, 4534, 2, 1231, 23, 1]
    print(a)

x()
```

如果想将下图的代码合并为一行，可以全选它们，然后按【Ctrl+Shift+J】即可合并代码为一行，还会自动补充代码

$$\begin{aligned}x &= 1 \\ y &= 1 \\ z &= 1\end{aligned}$$

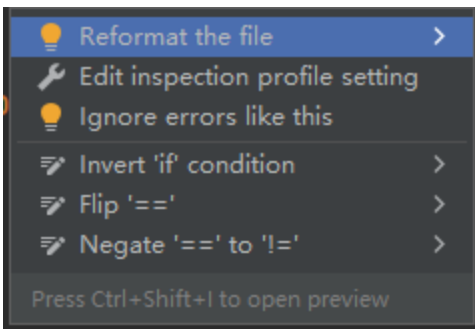
```
x=1; y=1; z=1
```

1/20

当出现黄色波浪号时，可以在对应代码出按下【Ctrl + Enter】进行修正代码的操作



按下后，会有多种选择供你修正，包括:格式化代码，忽略该警告，自动修改代码等



例如选择：【invert 'if' condition】会自动改成下面的代码：

```
def test(x):  
    if 1 != 1:  
        return  
    pass
```

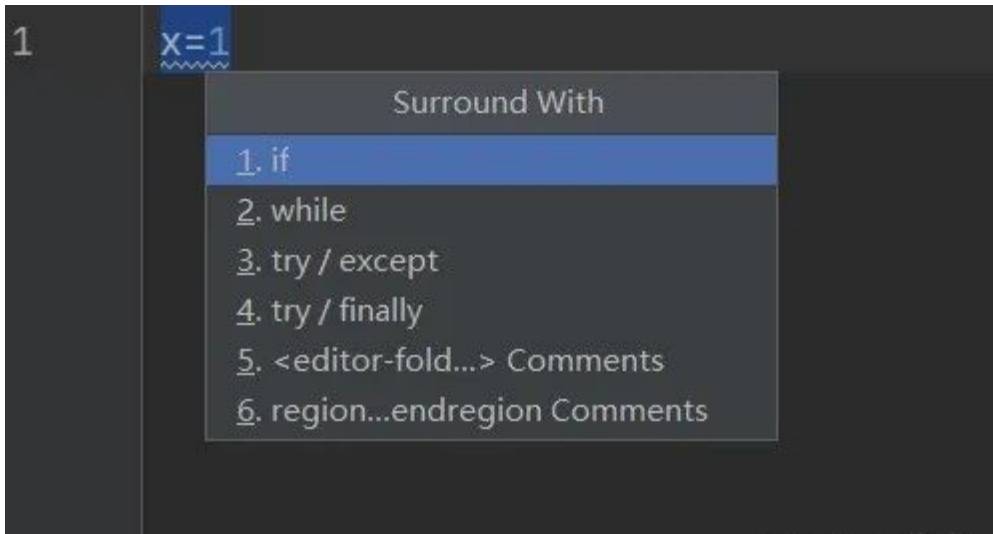
4 .包装代码【Ctrl+Alt+T】

我们可以快速的为输入的代码添加if、while、捕获异常等条件

例如有如下代码

```
x=1
```

我们选中该代码后按下【Ctrl+Alt+T】会弹窗让我们选择要包装的条件，现在我们选择【try/except】



包装效果

```
try:
    x=1
except:
    pass
```

5. 快速注释/取消注释【Ctrl+/】

如果我们想注释一部分代码可以选中对应的代码，并按下【Ctrl+/】

```
def show_text(text,a):
    a+=1
    print(text,a)
```

注释结果

```
# def show_text(text,a):
#     a+=1
#     print(text,a)
```

再次按下【Ctrl+/】则会取消注释

6. 向右缩进一个制表位【Tab】

python对代码的缩进要求很严格，下面的代码运行是会报错的！

```
def test():  
y = 1  
y += 1  
print(y)
```

但一行一行的去调整缩进非常难受，效率很低！这时候可以选中需要缩进的代码，按下【Tab】即可

效果

```
def test():  
    y = 1  
    y += 1  
    print(y)
```

7. 向左缩进一个制表位【Shift + Tab】

同上

8. 在上方插入新行【Ctrl + Alt + Enter】

如果想在下面代码 `a+=1` 的上方插入空行的话，可以点击到 `a+=1` 这行，然后按下【Ctrl + Alt + Enter】，则会在其上方新插入一行

```
def show_text(text,a):  
    a+=1  
    print(text,a)
```

效果

```
def show_text(text,a):  
  
    a+=1  
    print(text,a)
```

9. 在下方插入新行【Shift + Enter】

同上

10. 上下移动选中代码【Alt + Shift + 上、下键】

如果我们想将下面代码的 `a=1` 移动到 `print('click')` 上方，可以在 `a=1` 的所在行按下【Alt + Shift + 上】将其移动

```
def click(path):  
    print('click')  
    a = 1
```

效果

```
def click(path):  
    a = 1  
    print('click')
```

向下移动则按【Alt + Shift + 下】即可！

11. 上下移动选中方法体【Ctrl + Shift + 上、下键】

如果我们想将下面的 `send` 方法移动到 `click` 方法的上方，可以在 `send` 方法名这行(def 所在行)按下【Ctrl + Shift + 上】即可

```
def click(path):  
    print('click')  
  
def send(path):  
    print('send')
```

效果

```
def send(path):  
    print('send')  
  
def click(path):  
    print('click')
```

向下移动则按【Ctrl + Shift + 下】即可！

12. 复制代码【Ctrl + D】

如果我们想复制一行代码，可以在相应代码行按下【Ctrl + D】

```
x=y=z=1
```

效果

```
x=y=z=1
```

```
x=y=z=1
```

也可以选中多行代码进行复制

```
def show_text(text,a):  
    a+=1  
    print(text,a)
```

效果

但需要自己换行

```
def show_text(text,a):  
    a+=1  
    print(text,a)def show_text(text,a):  
    a+=1  
    print(text,a)
```

13. 折叠代码【Ctrl + -】

想折叠下面的代码的话，可以选中代码再按下【Ctrl + -】

```
def show_text(text,a):  
    a+=1  
    print(text,a)
```

效果

```
def show_text(text,a):...
```

14. 展开代码【Ctrl + +】

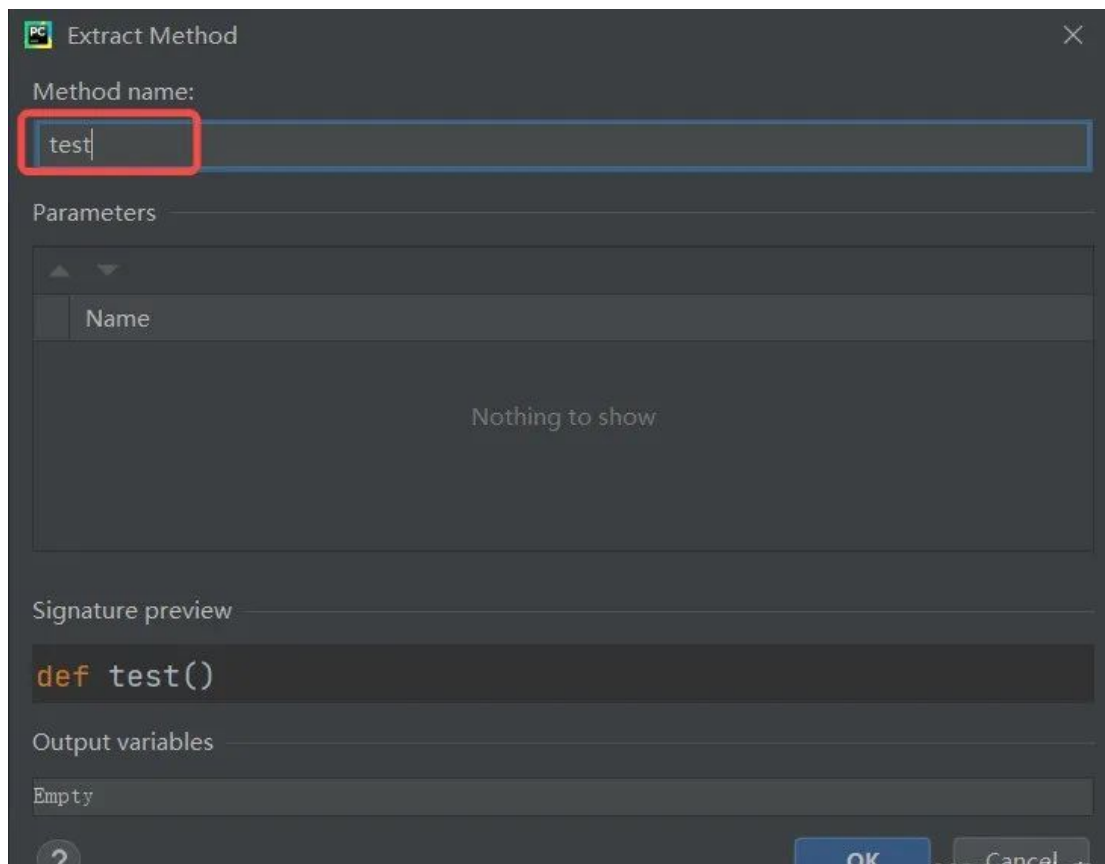
同上

15. 将代码抽取为一个方法【Ctrl + Shift+M】

如果想将如下的代码，写到一个方法中的话，可以选中代码并按下【Ctrl + Shift+M】

```
y=1  
y+=1  
print(y)
```

然后重命名方法，再点击【ok】

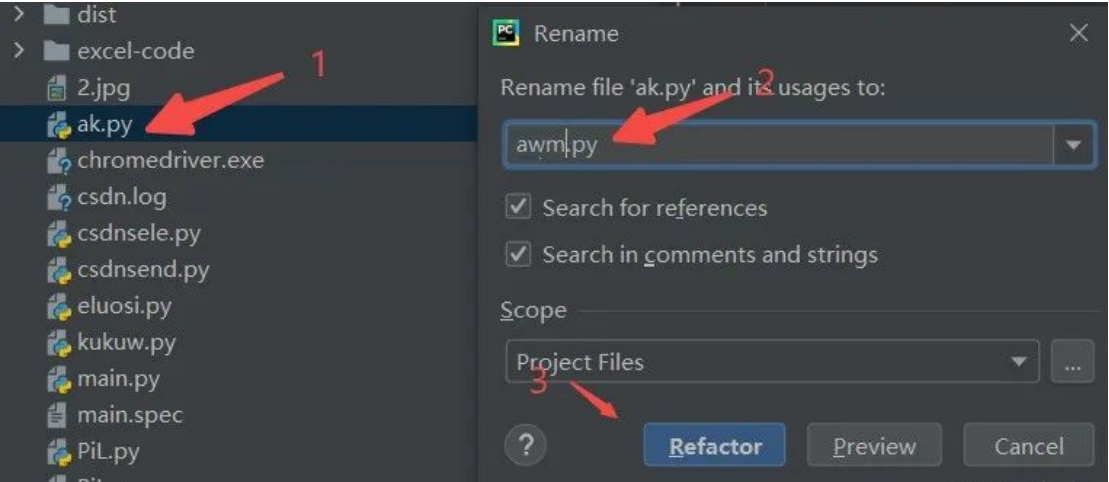


效果

```
def test():  
    global y  
    y = 1  
    y += 1  
    print(y)  
  
test()
```

16. 重命名文件【Shift+F6】

需要重命名文件名时，可以选择对应文件按下【Shift+F6】，再输入框输入新的文件名再点击【Refactor】即可

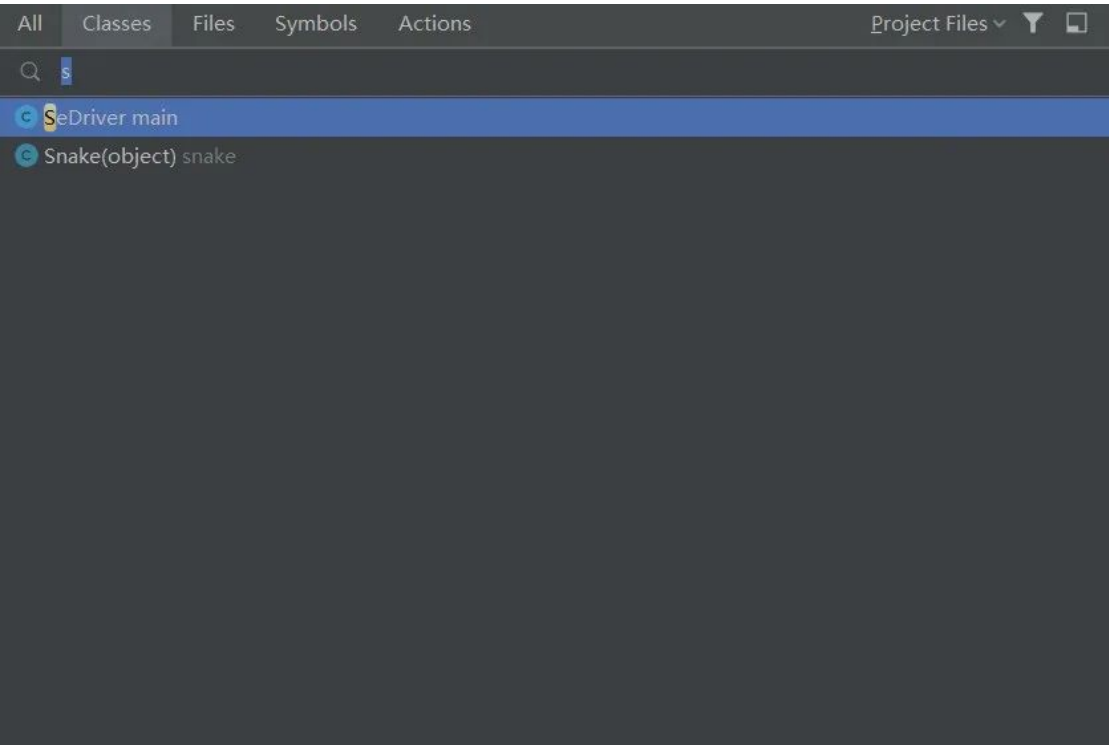


效果



17. 查找类被引用的地方【Ctrl+N】

按下【Ctrl+N】输入类的关键字，就可以看到被引用的类，点击对应的条目可跳转到对应文件



18. 查找/全局查找【Ctrl+F / Ctrl + Shift+F】

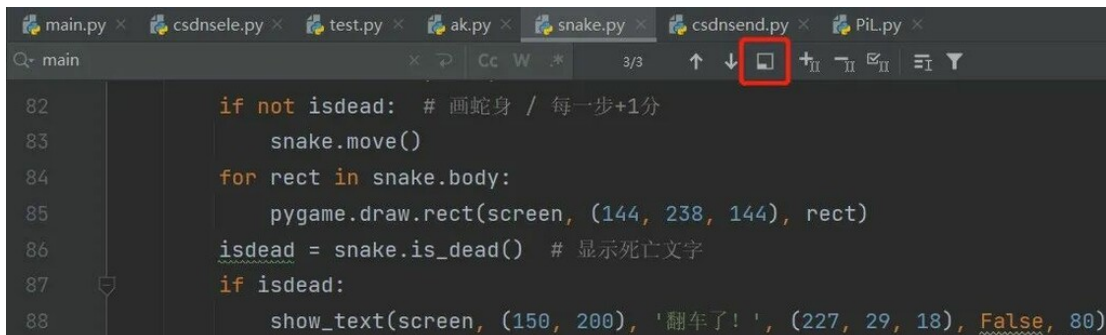
当前文件的查找可以按下【Ctrl+F】并输入要查找的关键字就会高亮包含关键字的代码



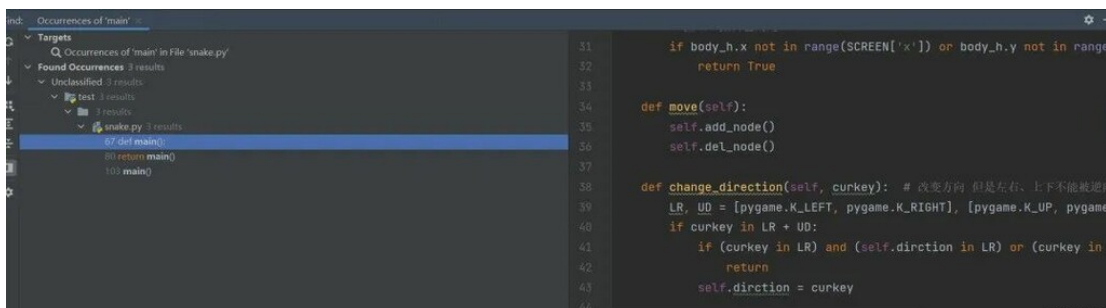
点击图中红款的箭头，可以逐行查看包含关键字的代码；另外，按下【Shift + F3】或【F3】也可以实现！



点击红框中的窗口图标，可以打开TOOL窗口进行多窗口查询



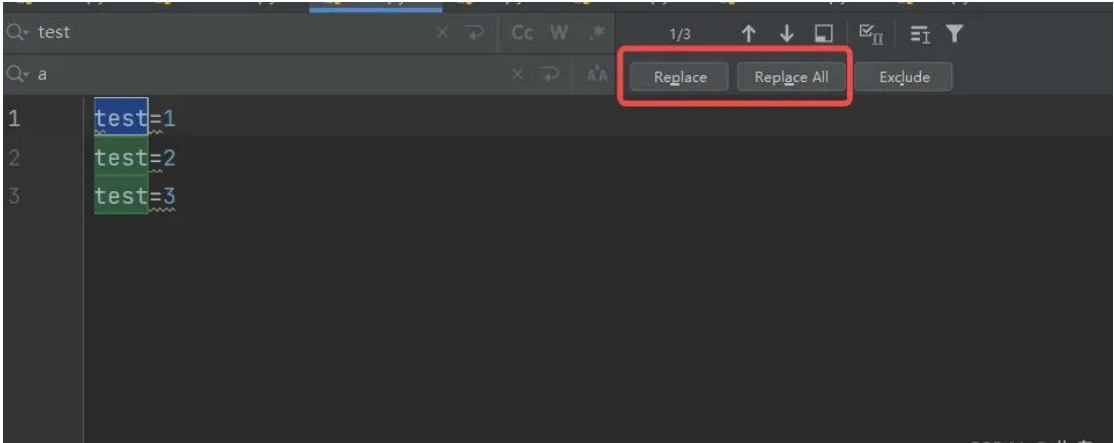
效果



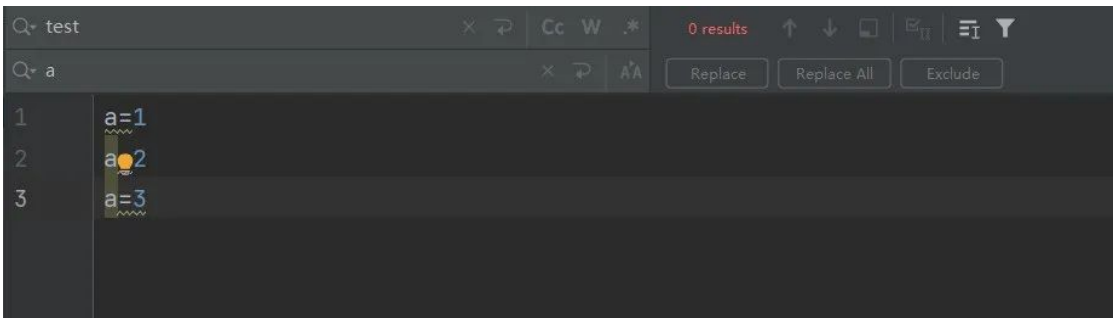
全局查询的话按下【Ctrl + Shift+F】即可！

19. 替换/全局替换【Ctrl+R / Ctrl + Shift+R】

当前文件的替换可以按下【Ctrl+R】并在第一栏输入要替换的关键字就会高亮包含关键字的代码，第二栏输入要替换为的关键字，在按下【replace】或【replace All】（替换全部）

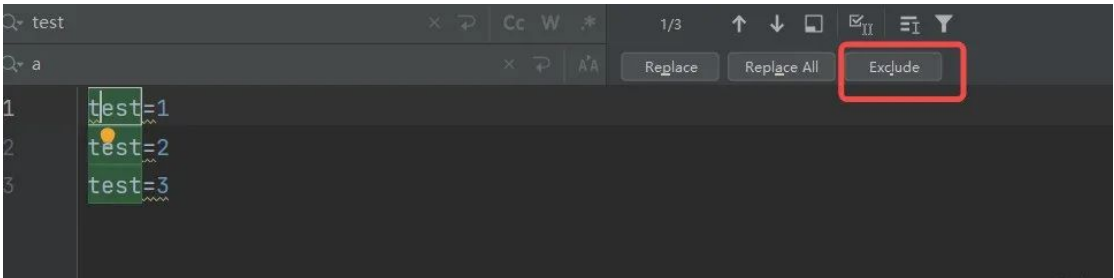


效果

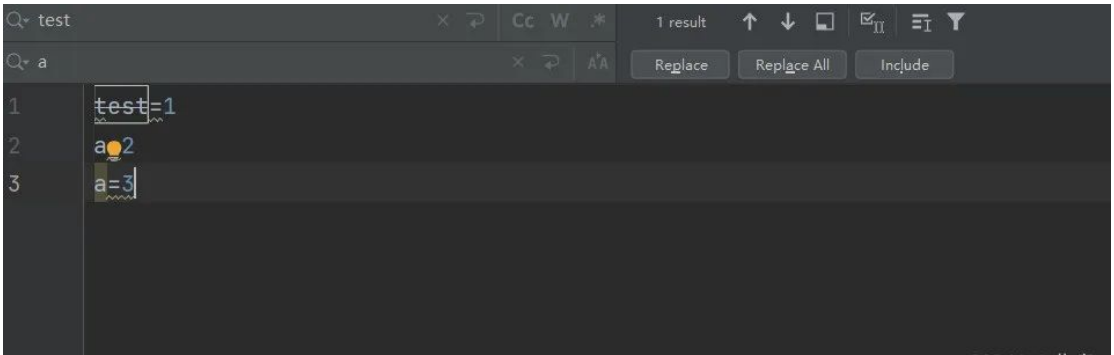


下图红框中的

【exclude】点击的话，会排除选中的该代码，只替换其他代码



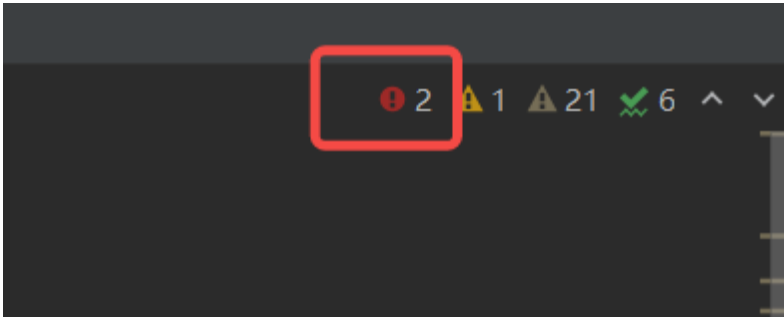
效果



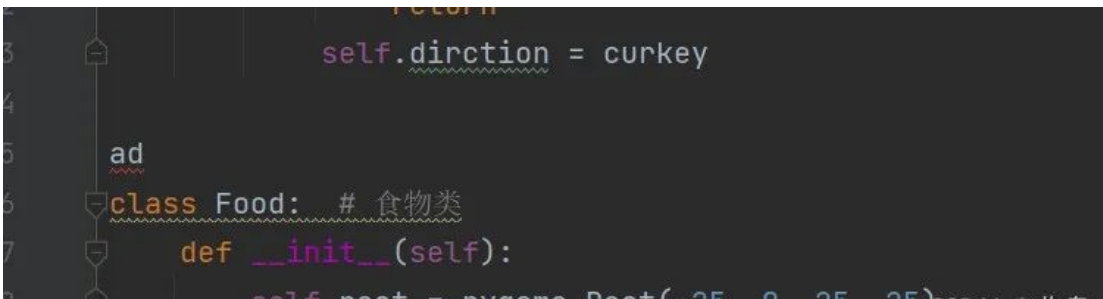
全局替换的话按下【Ctrl + Shift+R】即可！

20. 快速跳转报错的代码【F2】

当出现代码报错的时候，可以按下F2快速跳转到报错的代码处



效果

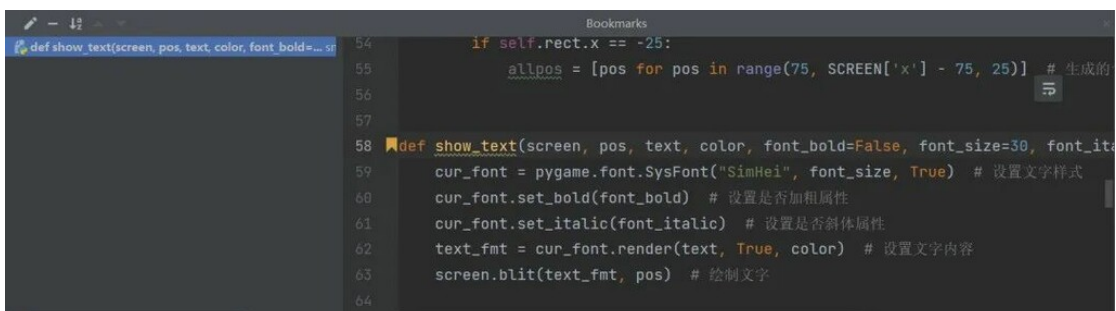


21. 定义一个书签【F11】

在相应代码处按下【F11】可以将其定义为一个书签



再按下【Shift+F11】，可以查看书签对应的代码



22. 代码小写转大写【Ctrl + Shift+U】

如果想将下面的代码转为大写，可选中代码后按下【Ctrl + Shift+U】

```
product_nama_dict={}
```

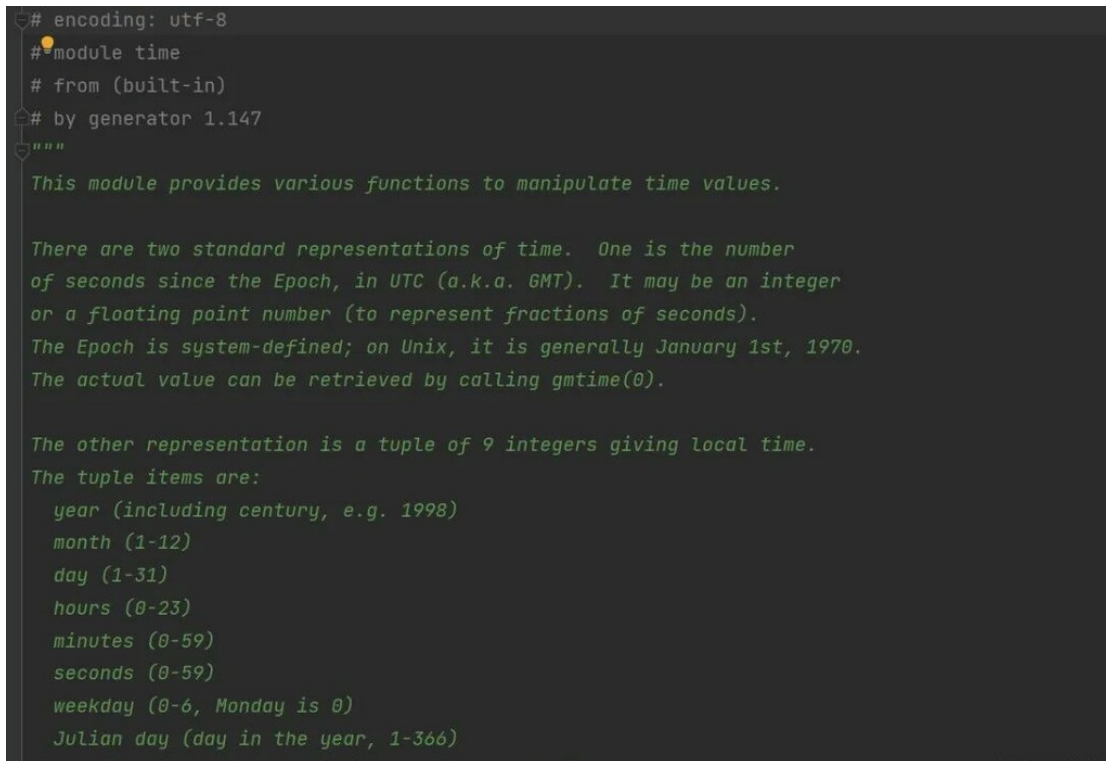
效果

```
PRODUCT_NAMA_DICT={}
```

23. 进入一个方法【Ctrl + B / Ctrl + 鼠标左键】

如果想进入time模块的方法中去，可以选中【time】再按下【Ctrl + B】，或者按下【ctrl+鼠标左键】

效果



```
# encoding: utf-8
# module time
# from (built-in)
# by generator 1.147
"""
This module provides various functions to manipulate time values.

There are two standard representations of time. One is the number
of seconds since the Epoch, in UTC (a.k.a. GMT). It may be an integer
or a floating point number (to represent fractions of seconds).
The Epoch is system-defined; on Unix, it is generally January 1st, 1970.
The actual value can be retrieved by calling gmtime(0).

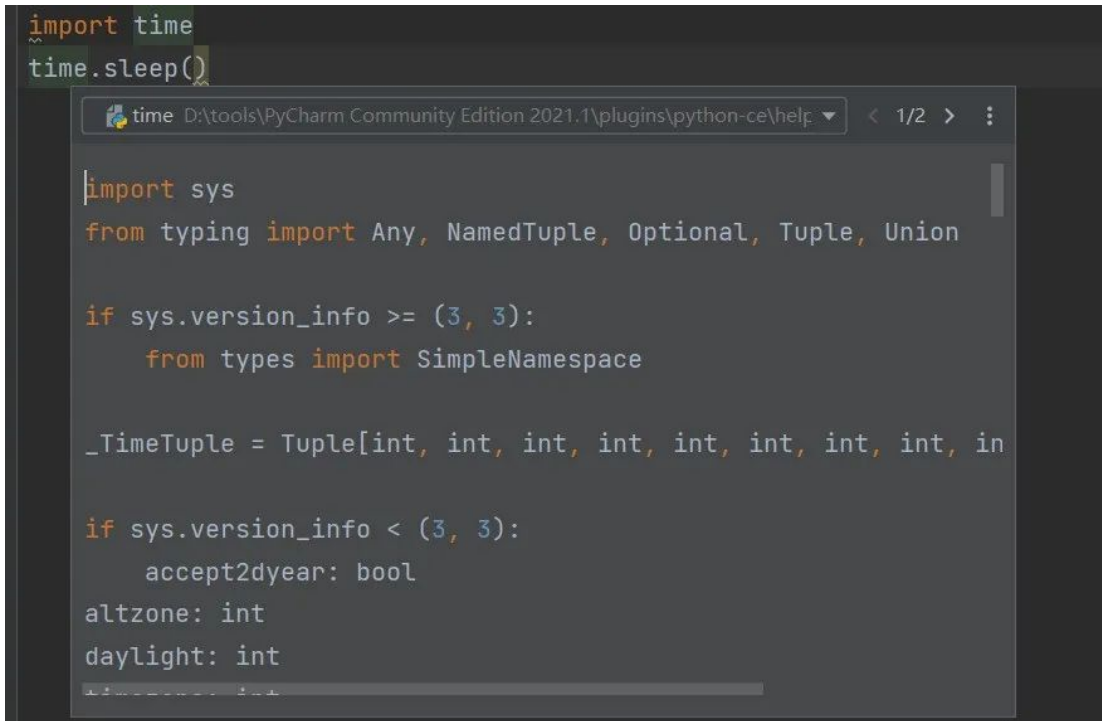
The other representation is a tuple of 9 integers giving local time.
The tuple items are:
    year (including century, e.g. 1998)
    month (1-12)
    day (1-31)
    hours (0-23)
    minutes (0-59)
    seconds (0-59)
    weekday (0-6, Monday is 0)
    Julian day (day in the year, 1-366)
    DST flag (1 if DST is active, 0 otherwise)
"""
```

24. 快捷查看方法的实现(源码)【Ctrl + Shift + I】

如果我们想看【time】是如何实现的，可以选中并按下【Ctrl + Shift + I】

```
import time
time.sleep()
```

效果



25. 查看文档描述【Ctrl + Q】

如果我们想看【time】的文档，可以选中并按下【Ctrl + Q】

```
import time
time.sleep()
```

效果

```
Module time

This module provides various functions to manipulate time values.
There are two standard representations of time. One is the number of
seconds since the Epoch, in UTC (a.k.a. GMT). It may be an integer or a
floating point number (to represent fractions of seconds). The Epoch is
system-defined; on Unix, it is generally January 1st, 1970. The actual value
can be retrieved by calling gmtime(0).
The other representation is a tuple of 9 integers giving local time. The tuple
items are:

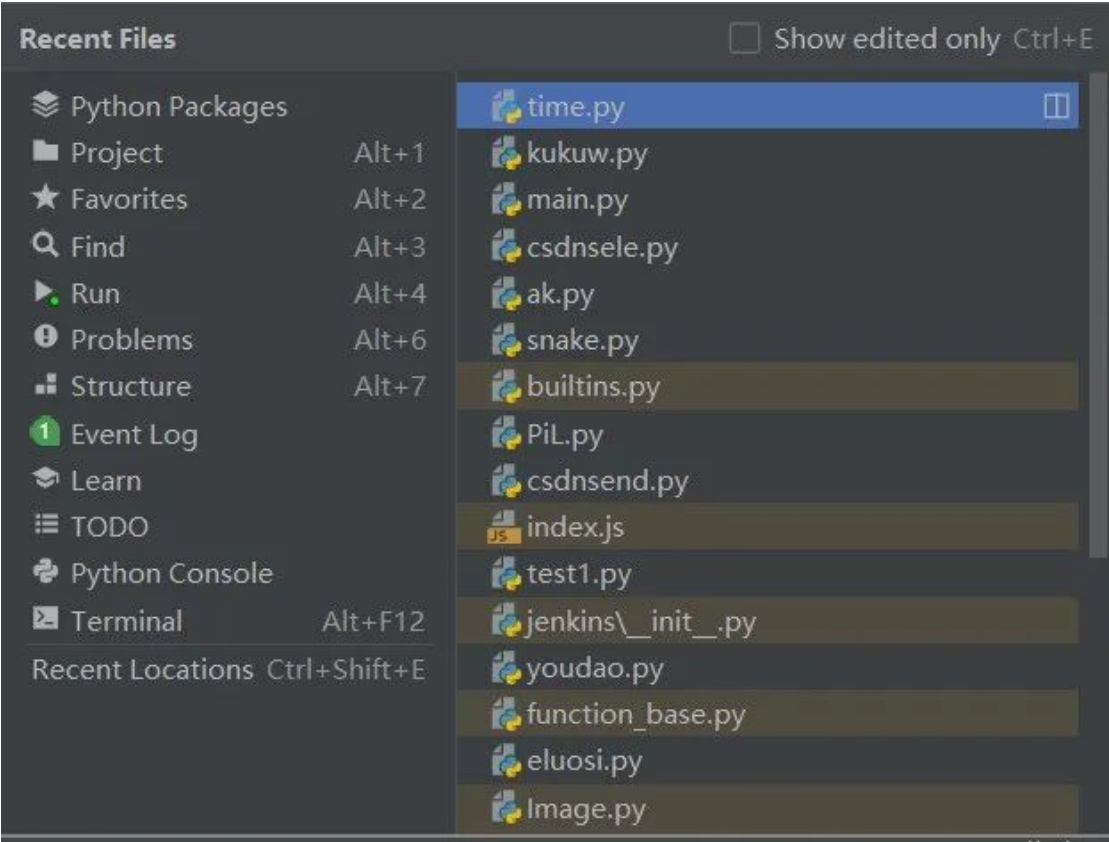
year (including century, e.g. 1998) month (1-12) day (1-31) hours (0-23)
minutes (0-59) seconds (0-59) weekday (0-6, Monday is 0) Julian day (day in
the year, 1-366) DST (Daylight Savings Time) flag (-1, 0 or 1)
If the DST flag is 0, the time is given in the regular time zone; if it is 1, the
time is given in the DST time zone; if it is -1, mktime() should guess based
on the date and time.

Scope: Non-Project Files
Size: 3.9 kB
Type: PythonStub
Modified: 2021/4/6 20:59
Created: 2021/4/6 20:59
📁 < Python 3.9 >
`time` on docs.python.org ↗
```

26. 查看文件中的方法【Ctrl + F12】

按下【Ctrl + F12】可以看该文件中有哪些方法、类

27. 最近编辑的文件列表【Ctrl + E】



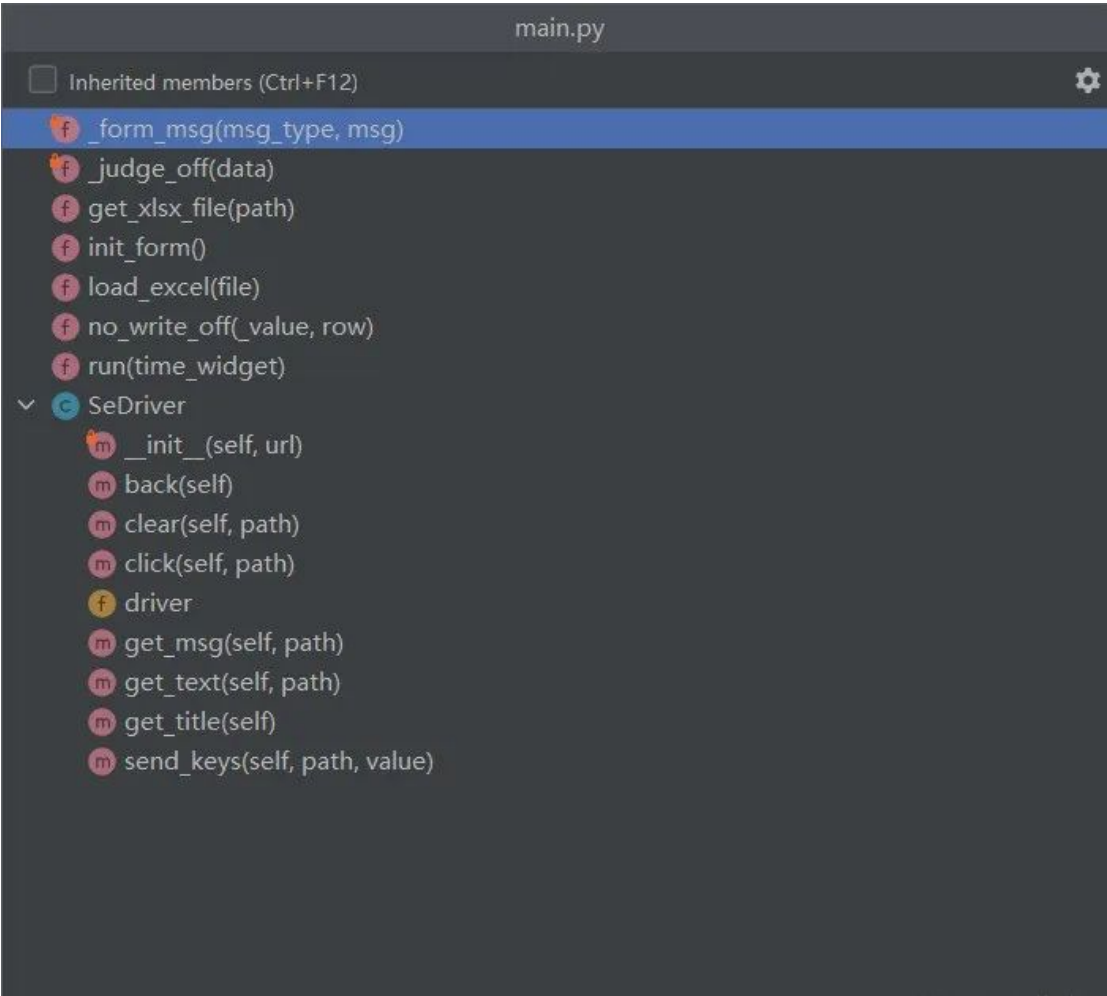
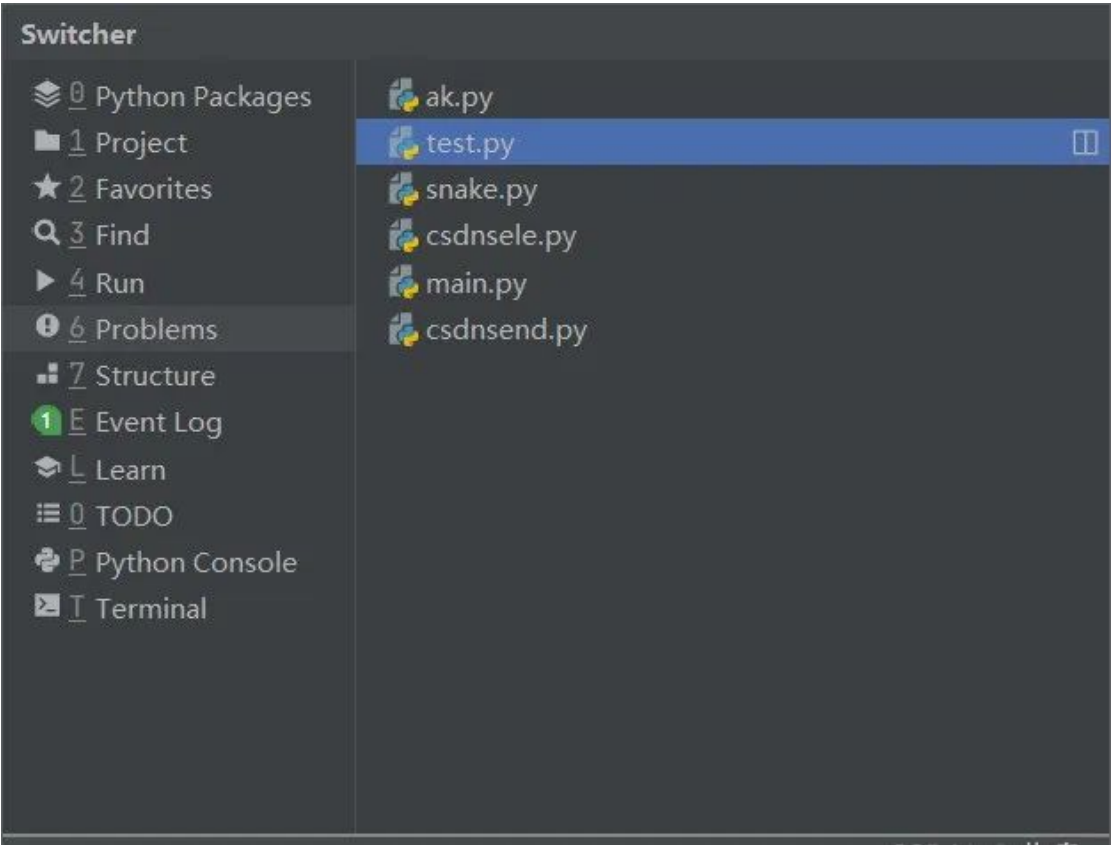
28. 快捷运行代码【Shift + F10】

快捷运行当前文件代码

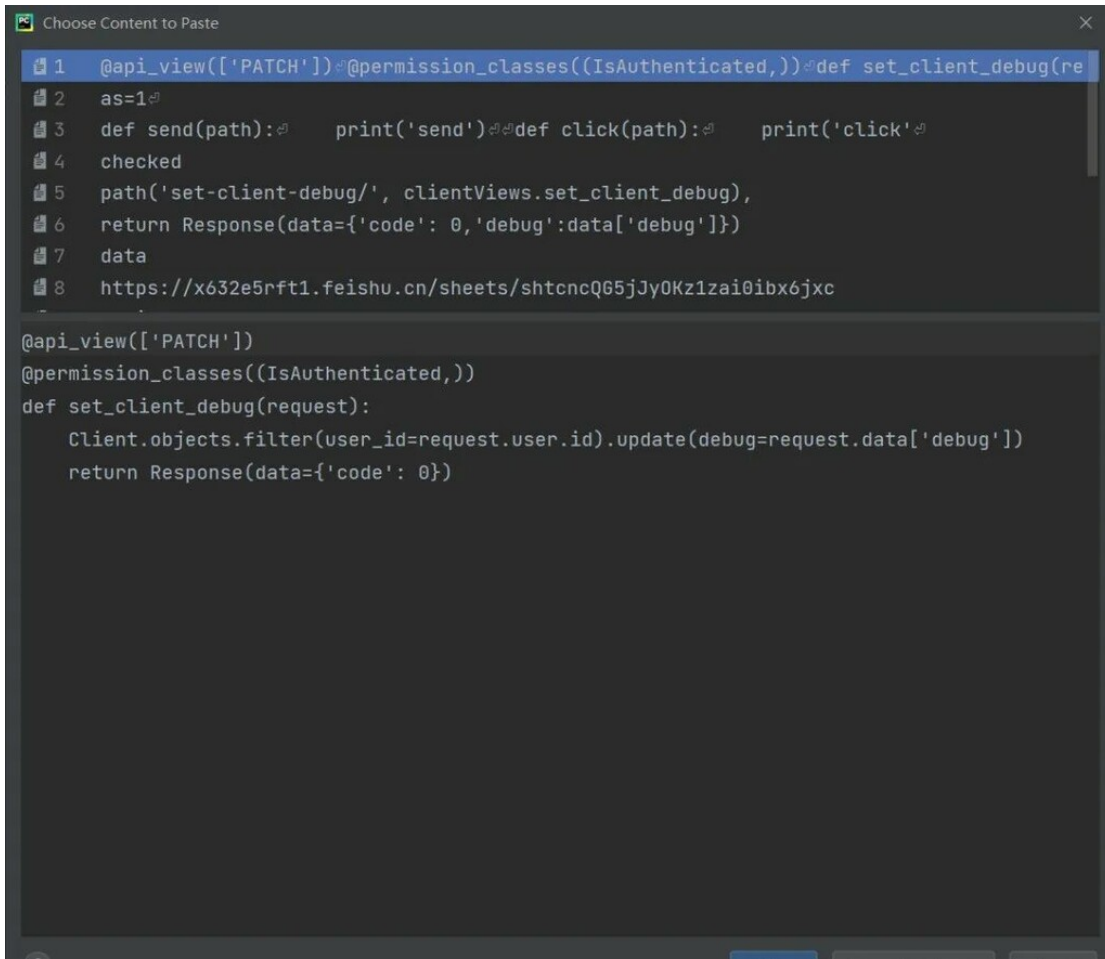
29. 快捷调试代码【Shift + F9】

快捷调试当前文件代码

30. 快捷切换视图/目录【Ctrl + Tab】



按下【Ctrl + Shift+ V】可以查看历史的复制粘贴记录



选择任意一行可以将它恢复回来

```
@api_view(['PATCH'])
@permission_classes((IsAuthenticated,))
def set_client_debug(request):

Client.objects.filter(user_id=request.user.id).update(debug=request.data['debug'])
    return Response(data={'code': 0})
```

35. 将光标移动到方法体或循环的开始【Ctrl + {】

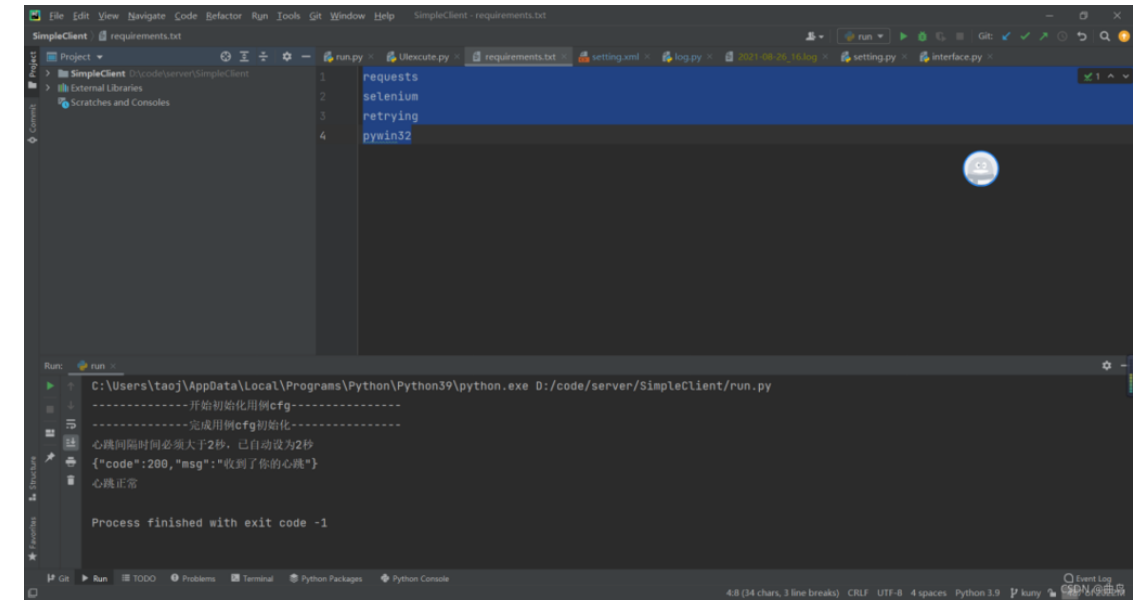
当你的方法或循环体很长的时候，可以按下【Ctrl + {】回到函数或循环头

36. 将光标移动到方法体或循环的结束【Ctrl + }】

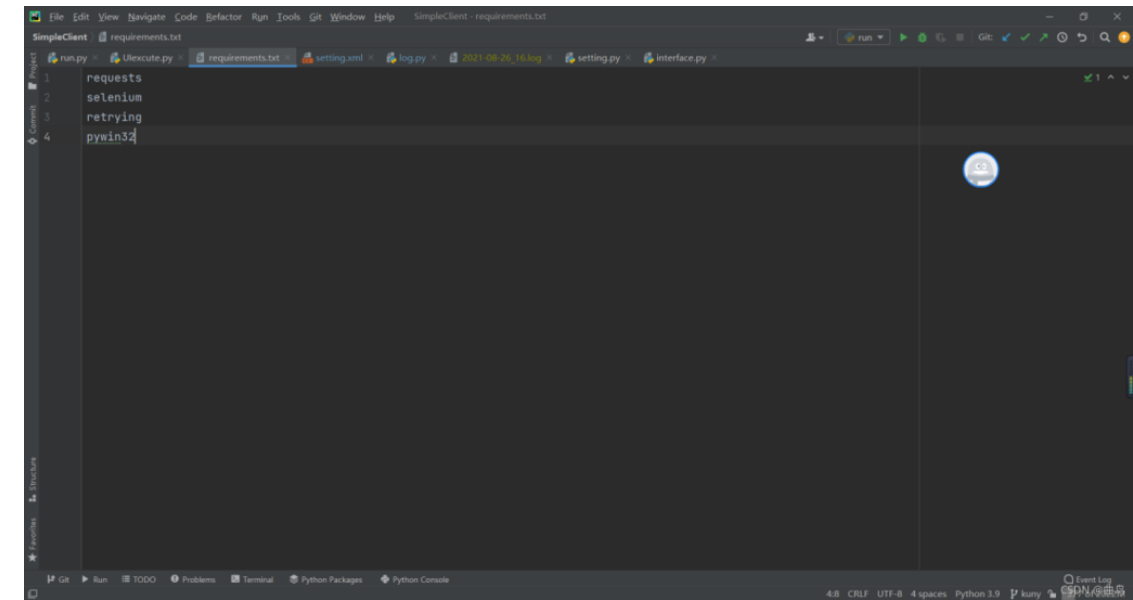
当你的方法或循环体很长的时候，可以按下【Ctrl + }】回到函数尾或循环尾

37. 最大化编辑代码窗口【Ctrl + Shift + F12】

当我们打开了多窗口的时候，影响了代码编辑体验的时候，如下图所示，可以按下【Ctrl + Shift + F12】隐藏其他窗口



效果



38. 快捷添加代码【Ctrl + J】

按下【Ctrl + J】可以快速添加代码

compd	Dict comprehension
compdi	Dict comprehension with 'if'
compg	Generator comprehension
compgi	Generator comprehension with 'if'
compl	List comprehension
compli	List comprehension with 'if'
comps	Set comprehension
compsi	Set comprehension with 'if'
iter	Iterate (for ... in ...)
itere	Iterate (for ... in enumerate)
main	if __name__ == '__main__':

例如添加 `if __name__ == '__main__':`，点击上图的【main】即可：

```
if __name__ == '__main__':
```