字段	说明	code
sublime.view	把整个文件的缓存，抽象成一个view。其实就是得到了文件的缓存	
id()	当前文件buffer对应的id	self.view.id()
buffer_id()	buffer对应的id	self.view.buffer_id()
is_primary		
set_name()	给当前的buffer设置一个名字	self.view.set_name('aaaa')
name()	当前buffer的名字	self.view.name()
is_loading()	直接运行运行失败，只有正在从硬盘中加载，加载完毕后你就可以用了，他就返回ture了	self.view.is_loading()
is_dirty()	运行证明：如果你更改后没有保存，则返回的是true 当然，如果你保存了，返回的就是false	
is_read_only()	我觉的这个就可以顾名思义了	
is_scratch() I set_scratch(value)	是不是把它设置成草稿，（没试过）	
setting	返回一个关于视图的设置对象	
window()	返回一个关于这个视图的窗口	
size()	这个很简单，就是说在所写的文件中，有多少个字符	不管注销，不注销，返回的就是多少个字符
visible_region()	返回一个可见的区域（注是可见区域的region）【如果你拉到代码都没有，也就是说可见区域的region就剩下字符串了】	
a,b,xpos	a,b就是region的两个坐标，xpos默认的是当前广播说在的位置，也就是-1， 你可以为它指定个值，想让我们的光标出现在Region的哪个地方	
sel()	返回一个可选区域selection的引用，这个可真牛逼，就是可以操作选中的，把选中的区域转化整Region	1.获得当前光标位置 self.view.sel()[0].begin() 2.下面这行代码，代表一个黑乎乎的位置，附着在你给的区域内 self.view.sel().add(sublime.Region(1,5)) 3. 把光标移动到想要的位置 self.view.sel().clear() self.view.sel().add(sublime.Region(1,1))
rowcol(point)	这个很有意思，就是把一维转换成二维，把buffer中的offset也就是参数，返回一个行，和列的数	这里的行和列都是，从第0行开始的
substr(region)	就是打印出相应Region的字符串	print(self.view.substr(sublime.Region(1,3)))
substr(point)	当然就是打印出第几个point的字符了 所以这个就能和rowcol一起使用，找出字符所在的具体的行和列了	
insert(edit, point, string)	第一次见到的就是这个方法,向偏移（point）的具体位置添加字符	
erase(edit, region)	很简单，	
replace(edit, region, string)	和上边一样	
sublime.Region(2, 5)	这个就可以直接创建Region，从几到几的Region	
line(point)	当前行所在的区域，返回的是Region	真tm不是返回的行数，而是当期行所在的区域啊！！ 大括号对其，就指着它了
line(region)	以Region的区域计算出整行的Region	
full_line(point) full_line(region)	这两个和上两个一样，就是就是包括new_line的字符	
lines(region)	返回每一行的region！！！	print(self.view.lines(sublime.Region(0,400))) 【(0, 14), (15, 36), (37, 37), (38, 38), (39, 39), (40, 89)】
split_by_newlines(region)	目前和上面一样，但上面的可能是有顺序的。	
word(point)	给出个偏移量，把单词的区域找出来 可以和substr一起使用	
word(region)	line（Region）以region的区域，计算出整个整词的Region区域	
find_by_class(point, forward, classes, <separators>)	这个还是返回的是一共多少个buffer，不是是Region 但还是根据后边的掩码返回的	sublime.CLASS_WORD_START sublime.CLASS_WORD_END sublime.CLASS_PUNCTUATION_START 就表示标点，只不过多个标点也表示一个标点，标点中有空格不算，但是有回车算一个标点， sublime.CLASS_PUNCTUATION_END 目前测试得知“” 是一个标点 { （ 是一个标点 sublime.CLASS_SUB_WORD_START sublime.CLASS_SUB_WORD_END sublime.CLASS_LINE_START sublime.CLASS_LINE_END sublime.CLASS_EMPTY_LINE print(self.view.find_by_class(3, True, sublime.CLASS_PUNCTUATION_END)) 上边那行代码，等忘了的时候，可以直接运行，但不知道为什么返回一个整数
expand_by_class(point, classes, <separators>)	两个收获：1.把英语的of有学习了一遍 2.知道了这个方法的用法，和classify一起用	如果扩展后，是按照sublime.CLASS_PUNCTUATION_START扩展的的，不管是前，是后，是包括标点的！！ 所以想着把标点删除 而且已经在START的位置，就会抛出当前这个标点了
expand_by_class(region, classes, <separators>)	和上边的一样，只不过第一个参数传入的是Region	
find(pattern, start_point, <flags>)	这个很简单，返回正则表达是匹配的region	self.view.find(‘import',0) self.view.find("edit", 0,）这个是直接正则表达是找值，最好用，一找就能找到
find_all(pattern, <flags>, <format>, <extractions>)	查找	self.view.find_all(‘import') 返回：[(0, 6), (15, 21), (137, 143)] 是一个数组
text_point(row, col)	也挺清楚的，根据给出的坐标，得出这个坐标在buffer中的偏移位置	注意：列是字符数，如果碰到特殊字符，它就会随着特殊字符传递给你要做的事情 如 \n. 在说一遍，列是从当前行，算后面有多少个字符
set_syntax_file(syntax_file)	这个在周六周日中，搞明白了，就是告诉他不同的文件后缀，他就会显示样式，你看有时候如果要不定义html后缀，你就没法使用html tab	

is_loading()	返回的是bool，打开文件可能单开线程，故这里是看文件是否加载完毕，我们使用它	def run(self, edit):
		view = self.view.window().open_file('/private/etc/hosts')
		def set_syntax():
		if view.is_loading():
		sublime.set_timeout_async(set_syntax, 0.1)
		else:
		view.set_syntax_file('Packages/INI/INI.tmLanguage')
		set_syntax()

extract_scope(4)	就是当前的点，属于哪个范围Region	self.view.extract_scope(4)
scope_name(point)	和上边的差不多	

字段	说明	code
match_selector(point, selector)	通过给出的point检查是否在seletor中	
score_selector(point, selector)	可以看出匹配几个	只感觉这个selector是个字符串，但又能使source.python
open_file(file_name, <flags>)	后边是从sublime text 3 forum 中找到的例子	view = self.view.window().open_file('c:/windows/system32/drivers/etc/hosts')
find_by_selector(selector)	返回匹配selector所有的region	
show(location, <show_surrounds>)	简单至极，展示到相关行(验证成功)	
show_at_center(location)	就是把该显示的在中间显示	
visible_region()	也是故名司仪	
viewport_position()	目前测得就是返回当前页面，可见部分右上角的那个列，注意：【0】代表第几列 【1】代表第几行	pos = v.viewport_position() v.show(point) new_pos = v.viewport_position() if abs(new_pos[0] - pos[0]) <= 1.0 and abs(new_pos[1] - pos[1]) <= 1.0: v.set_viewport_position((new_pos[0], new_pos[1] + 1))
set_viewport_position(vector, <animate<)	上面的知道什么意思，这个也就没什么问题了	
viewport_extent()	可以把sublime窗口缩小，就能知道他的意思了。可以说就是表示当前窗口的长和宽	
layout_extent()	viewport代表的是代码区域 layout代表的是软件区域	
text_to_layout(point)	根据给出的offset，得到当前点的layout右上角	
layout_to_text(vector)	与上边相反	
window_to_layout	可以直接用后边的代码试出来，不过多考虑	a,b = self.view.window_to_layout(self.view.text_to_layout(0))
window_to_text(vector)	一样	
line_height()	行高 多少个像素	
em_width()	字符宽度。	还是根据字符大小而改变
add_regions(key, [regions], <scope>, <icon>, <flags>)	不说了，后边的代码说明一切	reg = sublime.Region(0, 4) self.view.add_regions("key",[reg],"comment","dot")
get_regions(key)	我估计就是根据key得到所给的region,验证了	
erase_regions(key)	还是那句话，就是删除region	
set_status(key, value)	sublime 的左下角添加的内容	
get_status(key)	和上一样	
erase_status(key)	和上一样	
command_history	返回command名字，参数，使用次数	(a,b,c) = self.view.command_history(0) print(a) print(b) print(c)
change_count()	就看本文件改变了多少	
fold([regions]) fold(region) unfold([regions]) fold(region)	跟我想象的一样，就是折叠起来	
symbols()	反正能打印出一堆来，但也不知道是什么	symbols()
encoding() set_encoding(encoding) line_endings() set_line_endings(line_endings)	看是什么编码 II 以及设置什么编码格式 II 及当前行的编码格式 II 和设置当前行的编码格式	
overwrite_status() set_overwrite_status(enabled)	返回overwrite的次数	
show_input_panel(caption, initial_text, on_done, on_change, on_cancel)	后边有例子	v = window.show_input_panel('test', '', None, None,None) print(v.settings().get('is_widget')) v.set_syntax_file('Python.tmLanguage') print(v.settings().get('is_widget'))

show_popup_menu(items, on_done, <flags>)	尽在后边的三个例子	1.self.view.show_popup_menu(['foo', 'bar'], None)
		2.class SomeCommand(sublime_plugin.TextCommand):
		def run(self, edit):
		items = ['Text', 'Another']
		self.view.show_popup_menu(items, self.on_done)
		
		def on_done(self, result):
		print(result)
		3. items = ['Text', 'Another']
		self.view.show_popup_menu(items, self.on_done)
		
		def on_done(self,edit):
		if(edit == 1):
		print('aaa')

show_popup(content, <flags>, <location>, <max_width>, <max_height>, <on_navigate>, <on_hide>)	上边一搞明白，下边就so ease了	self.view.show_popup('suoyong',sublime.COOPERATE_WITH_AUTO_COMPLETE,2,200,200,None,None)
update_popup(content)	就是更新，上边出现的内容	self.view.show_popup('suoyong',sublime.COOPERATE_WITH_AUTO_COMPLETE,2,200,200,None,None) self.view.update_popup('zhaosongdi')
is_popup_visible()	猜都能猜到，已验证	
hide_popup()	和上边一组	
is_auto_complete_visible()		
sublime.Selection Class		
add(region)	太有意思了在sublime中所有的selections其实用的都是鼠标选中的selection，根本就没办创建一个selection，所以，如果你要使用selection只能使用self.view.sef()创建， 在这里最需要注意的就是selection里面的是region的数组，不是单独的一个region	1. b = self.view.sel().contains(sublime.Region(2,7)) print(b) 这个代码的意思是：你用鼠标拖动，选中一块区域，然后判断你拖动的区域是否有contains后边的区域，如果有则返回true，如果没有则返回false 2. self.view.sel().add(sublime.Region(0,9)) b = self.view.sel().contains(sublime.Region(2,7)) 此行代码的意思是，不用鼠标选中一块区域，看的是你add中的区域
clear()	因为是数组，clear()清除的就是所有的Region	
add_all(regions)	还是因为selection是数组，就可以以数组的形式加入region	

字段	说明	code
subtract(region)	如果你要知道subtract是什么意思，就明白了，其实就是减去selection数组中的一个region	
sublime.Region		
begion() **end()** **size()** **empty()**	都能明白	
cover(region)	返回一个横跨这个region和参数region，返回整个的一个region	
intersection(region)	两个region交互，返回交互的中间部分	
intersects(region)	看两个region是否有交集，有就返回true，没有就false	
contains(region)	一个region是否包含另一个region，或者看point在没在这两个region之中，有就true，没就false	
contains(point)		
Phantom	https://forum.sublimetext.com/t/dev-build-3118/21270	
SUBLIME WINDOW		
id()	还是通过self.view.window() 返回的window啊	self.view.window().id()
new_file()	简单粗暴，直接创建了一个view，	
open_file(file_name, <flags>)	也是简单粗暴，直接输入相对路径就可以	
find_open_file(file_name)	在已经有的Tab中找到相应的file_name	
active_view()	返回当前活动的窗口	
active_view_in_group(group)		
num_groups()	1.是有几个，	a = self.view.window().num_groups()
views()	多少个view，注意：返回的是一个数组	
active_group()	返回当前group的编号，其实挺有用的	

active_view_in_group(group)	自己多说两句group，经测试发现，其实就是sublime可以同时打开两个文件，这两个文件不就是组成一个group吗，但注意这两个文件执行哪个啊？ 我们发现，其实就是：后保存哪个，输入run的时候，就会先执行哪个。
	在这里还知道两个知识点：1，就是view就是一个tab。
	2. group呢，就是一个窗口，当然，一个窗口可以有多个group
	所以说这个方法是active_view_in_group(group)注意是view
		a = self.view.window().active_view_in_group(1/0)//1或0 下边的打印是不一样的
		# a = self.view.window().views()
		print(a.substr(sublime.Region(0,9)))

views_in_group(group)	这个就好说了，就是给出的group中有多少个view，注意返回的是[view]!!!	
focus_group(group)	你看上边第二行，还说谁后保存，就会run哪个了，你看这个，就不用保存了，就能run哪个，哈哈哈~~~	

focus_view(view)	找到group中个哪个view	import sublime
		import sublime_plugin
		
		class MoveTabCommand(sublime_plugin.WindowCommand):
		def run(self):
		a = self.window.views()
		self.window.focus_view(a[0])
		注意：一定要有indent，否则就会报indent错误

get_view_index(view)	很简单，乏善可陈	
set_view_index(view, group, index)	move the view to the give group with index	
status_message(string)	在状态栏中，展示一些信息 关键是左下角是状态栏	
is_menu_visible()		
set_menu_visible(flag)		
is_sidebar_visible()	边栏是否显示	
set_sidebar_visible()	设置是否显示边栏	set_sidebar_visible(False) set_sidebar_visible(True)
get_tabs_visible()		
set_tabs_visible(flag)	是否显示上面的Tab行	
is_minimap_visible()		
set_minimap_visible(flag)	看小窗口是否显示，true false	
is_status_bar_visible()		
set_status_bar_visible(flag)	下面的是status bar 是否显示	
folders()	没有验证，但根据目前的经验可知，这个folders指定就是在边栏中，你打开的foloder	
project_file_name()	其实就是folder的名字，只不过一般情况下，folder就是项目	
project_data()	两个都没有试，感觉一式就明白	
set_project_data(data)		

show_quick_panel(items, on_done, <flags>, <selected_index>, <on_highlighted>)		import sublime
		import sublime_plugin
		
		class MoveTabCommand(sublime_plugin.WindowCommand):
		def run(self):
		
		self.window.show_quick_panel(['aaa','bbb'],self.onon)
		
		def onon(self,index):
		
		print(index)
		
		sublime.KEEP_OPEN_ON_FOCUS_LOST 表示鼠标离开也不关闭
		倒数第二个参数，代表默认情况下，选哪个

字段	说明	code
show_input_panel(caption, initial_text, on_done, on_change, on_cancel)	上边的quick panel其实就是我们平常command + alt + p 搞粗来的， 而我们下面的这个input pannel是下面出来的一个输入框，第一个参数caption是一个字符串，就是告诉你输入框是什么，而输入框中的内容，就可以当参数传入到后边的回调函数中	解释：在弹出的输入框中输入，可以在回调的函数中 运行。

create_output_panel(name, <unlisted>)	output_panel其实就是一个，输出框， 但注意，他会返回这个输出框的view，当然，就可以想这个输出框中输入东西，使用insert，见右边第二个例子	1.class MoveTabCommand(sublime_plugin.WindowCommand):
		def run(self):
		self.window.create_output_panel('sf_st3_output')
		self.window.run_command('show_panel', { 'panel': 'output.sf_st3_output' })
		
		2.class SfprintCommand(sublime_plugin.TextCommand):
		def run(self, edit):
		self.view.insert(edit, self.view.size(), 'hello')
		
		class SfCommand(sublime_plugin.TextCommand):
		def run(self, edit):
		self.panel = self.view.window().create_output_panel('sf_st3_output')
		self.view.window().run_command('show_panel', { 'panel': 'output.sf_st3_output' })
		self.panel.run_command('sfprint');

find_output_panel(name)	找到相应的panel，找到后就想干什么，干什么呗	
destroy_output_panel(name)	就是消除输入框呗！！！	
active_panel()	console 其实就是sublime自己个定义的输入框呗	
panels()	拿这个panels一看才知道，这里的输入框真多啊 到现在初步明白，输入框就是下边那个白色框，有自己个定义的的，也有sublime自己嵌入进去的	
lookup_symbol_in_index(symbol)	这个是找到整个项目中的字段，或者说是类名，下面的是找到打开文件中的类名或字段	
lookup_symbol_in_open_files(symbol)	这个还真等同于command + alt + r 就是找到整个打开文件中的symbol	
extract_variables()	不多说了，直接放回	packages, platform, file, file_path, file_name, file_base_name, file_extension, folder, project, project_path, project_name, project_base_name, project_extension. This dict is suitable for passing to sublime.expand_variables().
sublime.Settings		
get(name, <default>)		view.settings().get('syntax')
set(name, value)	和上边对应一-感觉像个全局变量，有这种感觉	
erase(name)		
has(name)		
add_on_change(key, on_change)	当有全局变量改变时，使用回调函数	
clear_on_change(key)	就是清除回调函数	
sublime_plugin.EventListener		

on_new(view)
on_new_async(view)
		import sublime, sublime_plugin
		class some_bug(sublime_plugin.EventListener):
		def on_new(self,view):
		print ('aa')

on_clone(view) **on_clone_async(view)**	关键是知道怎么克隆，使用下列命令： New View into File in the File	

on_load(view)
on_load_async(view)
	也就是说，当执行后边的这个代码使，他才会代用on_load这个监听	class MoveTabCommand(sublime_plugin.WindowCommand):
		def run(self):
		self.window.open_file('test.js')

on_pre_close(view) **on_close(view)**	一个是将要关闭，一个是已经关闭	
on_pre_save(view) **on_pre_save_async(view)**	save以前	
on_post_save(view) **on_post_save_async(view)**	save以后	
on_modified(view) **on_modified_async(view)**	终于找到质数教学的简单方式了，当改变的时候就会调用这个方法，这个就是为点击以后的铺垫	
on_selection_modified(view) **on_selection_modified_async(view)**	自己感觉就是只有改变固定的地方才会调用这个方法	
on_activated(view) **on_activated_async(view)**	就是focus哪个窗口就调用一下这个函数	
on_deactivated(view) **on_deactivated_async(view)**	失去焦点时调用	print (view.id()) //可以直接打印id的
on_hover(view, point, hover_zone)		
on_query_completions(view, prefix, locations)	这个是我学习node.js中，第一个想知道的方法，或者是觉得有用的方法。这次终于可以深入了解一下	1.要明白这个是回调函数，所以这里的参数也就是 view prefix locations都是当你点击时，自动赋给的！！！ 如： def on_query_completions(self,view, prefix, locations): print(locations) print(prefix) 2. https://forum.sublimetext.com/t/solved-possible-to-launch-custom-autocomplete-on-modified-or-include-word-separators-on-query-completions/18942/14 这个网站的确nb可以以后参考 3.这个方法主要看返回，当然可以返回空， 3.1：也可以返回：return [["me1", "method1()"], ["me2", “method2()"]] 只提示me1 按tab键可变成method1 （） 3.2：return [["me1\tmethod", "method1()"], ["me2\tmethod", "method2()"]] 不仅提示me1而且在提示栏的尾部，还提示method1 4.return [["fn", "def ${1:name}($2) { $0 }"], ["for", "for ($1; $2; $3) { $0 }"]] 这个就是按照格式提示了 5.现在大家看一下，知道下面这是个tuple就可以了 注意return后面是一个大括号，可不是中括号了 return ([["me1", "method1()"], ["me2", "method2()"]], sublime.INHIBIT_WORD_COMPLETIONS | sublime.INHIBIT_EXPLICIT_COMPLETIONS)
on_query_context(view, key, operator, operand, match_all)	这个可能就是按个键盘上的键，然后就能运行plug 这个命令可以对应下面绿色部分的keybind	1.搞了半天，就稍微搞懂了一点，就是默认的key肯能是enter，operand 等于什么就是执行设么 2.官方文档说的give context key我觉得就是下边这个链接里的东西 http://docs.sublimetext.info/en/latest/reference/key_bindings.html

字段	说明	code
on_text_command(view, command_name, args)	这个官方文档不是说有返回值吗 其实这个返回值就在这个方法的参数中，就是	def on_text_command(self,view, command_name, args): print(command_name) print(args) 在这个命令下，上下左右鼠标点击，都会有相应的,delect也有相应，就是键入字母的时候没有相应
on_window_command(window, command_name, args)	其实这个命令和上个命令正好相反，上个是监听view_command， 这个命令是监听window_command	只要运行window.run_command(‘’)的时候就会调用前面这个方法
on_post_text_command(view, command_name, args)	和上边第一个一样	
on_post_text_command(view, command_name, args)	和上边第二个一样	
sublime_plugin.ViewEventListener	上边那个listen是那个view都可以，当前这个listen是只能特定的view	
is_applicable(settings)	python真有类方法，这个就是类方法	a = sublime_plugin.ViewEventListener.is_applicable(self.view.settings()) print(a)
applies_to_primary_view_only()		sublime_plugin.ViewEventListener.applies_to_primary_view_only()

on_modified()	不说了，终于让我给试出来了，其实ViewEventListener和EventListener就二点不同，就是
	1，前者只服务于view层，后者服务于所有
	2. 最关键的是前者可以用于指定view的相应，而后者是所有都能相应。
	可以看后边的例子
		有ViewEventListener的文件
		import sublime, sublime_plugin
		class some_bug(sublime_plugin.ViewEventListener):
		@classmethod
		def is_applicable(cls,settings):
		syntax = settings.get('aaa')
		print(syntax)
		# return 1
		return syntax == 'sub'
		
		def on_modified(self):
		print(‘a')
		可以看到，在其中增加了is_applicable的类方法。
		设置setting的文件
		class AaCommand(sublime_plugin.TextCommand):
		def run(self, edit):
		self.view.settings().set(‘aaa','sub')
		看到了吧，这就能看出设置setting的用途了，还能看见好处了

其他都一样！！！		

sublime_plugin.ApplicationCommand	感觉就是对整个程序的操作，比如像后边，设置程序的配置文件
	这是直接使用sublime.run_command('example')来运行此命令
		import sublime, sublime_plugin
		class CustomSetSettingCommand(sublime_plugin.ApplicationCommand):
		def run(self, file, key, value):
		settings = sublime.load_settings(file)
		settings.set(key, value)
		sublime.save_settings(file)

sublime_plugin.TextCommand	可以通过self.view来得到这个类的view	
view.run_command()	在console中执行py文件	
run_command(string, <args>)	注意：不管是view还是，还是windows，我们需要知道的就是，run_command(的参数后面，要有参数)	
description()		

set_timeout()	延时多长时间运行	class SetTimeoutTestCommand(sublime_plugin.WindowCommand):
		def run(self):
		for x in range(1,10):
		sublime.set_timeout(lambda : print(x), 4000)

file_name()	当前文件的名字	self.view.file_name()
	保存就会执行，写入py的代码	kingkeithAug 17 Nevermind, I've managed it: Create a new tab, paste in the following: def test(): print('wow') print('hello') test() put the cursor after hello'). in the console: view.add_phantom("test", view.sel()[0], "Hello, World!", sublime.LAYOUT_INLINE) click the folding arrow on line 1 to collapse it, and ensure the caret is not on line 4: line 4 in the gutter shows the number 3 underneath the 4. tested in Windows on build 3120. CC @jps
	Edit objects 只能被TextCommands创建，不能被其他的东西创建，它的存在就为了整buffer中的内容。（API中单独有他的解释）	
右键用sublime 打开文件夹	1.在mac上打开automater 2.选择服务Service，点击选取 3.在右边的输入框中，输入shell 选择运行shell脚本（Run Shell Script) 4.把选择shell脚本拖入到左侧的窗口中 5.把sublime中显示包内容中的subl复制到窗口中，并在后面添加 -n $@ 6.command + 梅花 储存 并取名	
command + D(ctr + command + d)	可以多重选择，按一次选一次，一次多选	
sublime module		
set_timeout(callback, delay)		sublime.set_timeout(aa, 3000)
set_timeout_async(callback, delay)	不阻塞线程	
error_message(string)	这个error，能弹出个对话框	sublime.error_message('aaa')
message_dialog(string)	效果和上边那个error一样，使用也一样，只不过这个是发送的不是错误，而是消息	
ok_cancel_dialog(string, <ok_title>)	注意：这个是返回的bool值	sublime.ok_cancel_dialog('aaaa', ‘ddd') 如果给出第二个参数，则ok用第二个参数代替

字段	说明	code
yes_no_cancel_dialog(string, <yes_title>, <no_title>)	还是上边用法	只不过返回的是：sublime.DIALOG_YES, sublime.DIALOG_NO or sublime.DIALOG_CANCEL
find_resources(pattern)	找到这个资源，根据pattern	customCompletion_Files = sublime.find_resources ("*.custom-completions") file = customCompletion_Files[0]
load_resource(name)	加载这个文件，文件的名字	jsonString = sublime.load_resource (file)
load_binary_resource(name)	看文档吧	
decode_value(string)	把json串，变成对象	jsonValues = sublime.decode_value (jsonString) 注意：python访问JSON中的键值是这么访问的 fileExtensions = jsonValues["fileExtensions"] completionSeparator = jsonValues["separator"] jsonCompletions = jsonValues["completions"]
encode_value(string)	一个是吧JSON变成对象，一个是吧对象变成JSON	
expand_variables(value, variables)		aa = sublime.expand_variables("Hello, ${name}", {"name": "Foo"}) print(aa) 看到了吗，这么可以用 如果是字符串，必须是snippet的语法
load_settings(base_name)	看文档	根据下边的package_path()我们得知,base_name其实就会去package这个文件中去找
save_settings(base_name)	很简单，就是把setting保存	
windows()		print(sublime.windows())
active_window()	一样，当前活动的window	
packages_path()		print(sublime.packages_path()) 打印出：/Users/suoyong/Library/Application Support/Sublime Text 3/Packages
installed_packages_path()	只不过换成installed_packages_path()	installed_packages_path()
cache_path()	cache的文件	
sublime.get_clipboard()	这到挺有意思，告诉你现在复制的是哪个值	
set_clipboard(string)	与上相反	
score_selector(scope, selector)	区域比较，估计又是Region和sel（）的比较在这里就不实验了	

run_command(string, <args>)	注意这个就更ApplicationCommand	class ExampleCommand(sublime_plugin.ApplicationCommand):
		def run(self):
		print("aa")
		
		sublime.run_command('example')

log_commands(flag)	这个就是那个价值百万的问题	
log_input(flag)	遇到正则表达式，在搞这两个	发现价值百万的问题，中的flag也不过是True和False
log_result_regex(flag)		
version()	返回版本号	
platform()	返回平台	
arch()	返回cpu的参数	
keybindings	1.路径是user/.keymap，不同的操作系统，对应不同的文件 2.keybing最简单的是[{ "keys": ["ctrl+shift+n"], "command": "new_window" }, { "keys": ["ctrl+o"], "command": "prompt_open_file" }] 表示一个key对应一个，我们说的panel中，需要运行的命令，比如:view.run_command(‘aaa’) 及command后边的值就是我们说的aaa 3.最关键的是，我们的keybings还可以向里面传参数 { "keys": ["shift+enter"], "command": "insert", "args": {"characters": "\n"} } // 这个就是传的characters的参数等于\n 跟我想的一样，这个参数传个的是plug中，def run(self,view,characters) 这里的后边加一个characters 4.{ "keys": ["escape"], "command": "clear_fields", "context": [{ "key": "has_next_field", "operator": "equal", "operand": true }] } 这里context就是你可以给出一些条件，在某些时候，不让使用这个快捷键 这个网址给出了一些参数： http://docs.sublimetext.info/en/latest/reference/key_bindings.html。其实这个正对应on_query_context	
执行python	一保存就直接执行了	
		https://forum.sublimetext.com/t/run-command-vs-internal-api-and-the-future-of-sublime/12654
	值得一看	https://forum.sublimetext.com/t/api-suggestions/20640/60
	https://www.youtube.com/watch?v=PFP0oXNNveg	https://www.youtube.com/watch?v=r9LCwI5iErE
snipers	定义就是当你点击tab键的时候，sublime会自动为你补全 1.tools -> snepit 可以找到snepit	创建一个自己的snippets 1.tools -> developer -> new snippets 2.进入没有保存的sublime自己定义的文件的格式 <scope>source.python</scope> source.js source.python source.css text.html 3.设置输入什么内容后，点击tab补完内容 <tabTrigger>hello</tabTrigger> //输入hello 后，点击tab就会补完 <content><![CDATA[ ]]></content> 中的内容 4.如何输入复杂内容’$+’使我们的替换变量 4.1${1:cat} 默认的替代此 4.2${1/*//} 不让光标闪动

字段	说明	code

key binding	就是添加快捷键，但上来就一个收获，不是python老报indent错误吗，直接：
	editor -> line -> reindent
		如何创建快捷键
		1.Mac -> Preferences -> key binding (注在sublime 3中打开，直接是两个个key binding 组
		2.基本skelton:
		[
		{“key”:[“”],”command”:””}
		]
		3.millions dollas question: 如何知道command的名字
		在console窗口中，输入sublime.log_commands(True) //这个命令，就是不管你干什么，都能打印出相应的log。
		4.当然，skelton中，也可以增加参数，”arg”:{“file”:”/”} //见下面，宏中的第5点

Macros	这里的宏，特像魔兽世界中的宏，就是一系列的命令	如何设置宏： 1.按住command + Q 开始记录宏 2.按住command + Q 结束记录宏 3.tools -> playback Macro 在当前版本中，tools中对宏都有说明 4.tools -> save Macros 可以把宏保存在，user文件中，注意存储的格式是sublime-macro 5.通过key binding运行记录的宏 {“alt+enter”],”command”:”run_macro_file”,”args”:{“file”:”Packages/User/new-curly-line.sublime-macro”} 6.可以直接通过，sublime，你保存的宏文件，看看你各种的操作命令，其实这个文件很明了，就是一个keybinding里面执行的各种命令

Add Button		如何给菜单中的各种菜单添加按钮：
		1.在Packages目录中创建Main.sublime-menu文件
		2.在文件中写上
		[
		{
		"id": “edit", //表示在那个菜单中添加
		"children":
		[
		{
		"id" : “htmlcodeconverter", //随便取名字,
		"caption" : “飞雀", //菜单中显示的名字
		"command" : “example" //运行的命令
		}
		]
		}
		]
		3.我估计这个会和我的plugin包放在一起，下载下来后，会直接在菜单中添加

text theme and color scheme		如何设置 Mac -> preference -> color scheme
把光标在上中下移动	看这个网址 Move cursor to top/middle/bottom of visible lines	https://forum.sublimetext.com/t/move-cursor-to-top-middle-bottom-of-visible-lines/4586