Python期末作业报告

题目序号： 36 文本编辑器

学号：2016329621013 姓名: 高宗继

学号：2016330301072 姓名: 郑玉梅

学号：2016330301073 姓名: 朱小艳

学号：2016327130001 姓名: 白紫嘉

小组分工：

高宗继：代码实现（基础功能）
郑玉梅：资料搜集与程序设计
朱小艳：代码实现（语法高亮）、报告撰写
白紫嘉：代码实现（界面实现）

实验内容及要求：

- 用进sublime吗？希望你能够写一款单文件，但是高效、易用的文本编辑器，如果你用你自己的文本编辑器，来编写python，是多么有成就的一件事啊。
- 文本编辑器最好能够做到以下几点：
  - 多语言支持，比方说Python，C等；
  - 语法高亮、自动补足，如果能够自动提示语法，已定义过的变量或函数，那就更好了；
  - 至于界面，参考Sublime，或其他优秀的编辑器，只是外表，我们更注重内心。

设计思想和解决方案：

本次项目为设计一个代码编辑器，并实现代码高亮，首先要解决的就是python GUI编程的问题。python常用的GUI编程的包有PyQt5、wxpython和Tkinter等，三者相比之下选择了Tkinter，因为Tkinter虽然功能上来说比wxpython和PyQt5弱一些，但是相对来所比较简单，容易入门，其中的Text、Menu、Lebel、Font等控件用于编写代码编辑器也很方便，所以选择Tkinter来做GUI编程，实现代码编辑器的界面。其次要解决的问题是实现代码高亮等操作，我们选择使用pygments包来实现代码高亮，通过将键盘输入的代码分解为token，与pygments.tokne模型进行对比，如果有则设置特殊的颜色，不同类型的token设置不同的颜色，实现代码高亮。在实现基本功能之余，添加了部分其他功能，实现了文件的新建、打开、保存、另存等，编写代码的时候添加了撤销、重做、复制、粘贴、剪切、回到文件顶部和底部的快捷键，并实现了修改字体的功能。

源代码：

Tkinter 是使用 python 进行窗口视窗设计的模块。Tkinter模块("Tk 接口")是Python的标准Tk GUI工具包的接口。作为 python 特定的GUI界面，是一个图像的窗口，tkinter是python 自带的，可以编辑的GUI界面，我们可以用 GUI 实现很多直观的功能，比如想开发一个计算器，如果只是一个程序输入，输出窗口的话，是没用用户体验的。所有开发一个图像化的小窗口，就是必要的。

In [10]:

```
import tkinter as tk
import tkinter.filedialog
import traceback
import tkinter.ttk as ttk
import tkinter.font
from tkinter import messagebox
from tkinter import colorchooser
from pygments import lex  #Pygments提供了十多种高亮样式的方案
from pygments.lexers import PythonLexer
```

Tkinter支持一些核心的窗口部件，现将使用的窗口部件类简要描述如下：
Button：一个简单的按钮，用来执行一个命令或别的操作。
Canvas：组织图形。这个部件可以用来绘制图表和图，创建图形编辑器，实现定制窗口部件。
Label：显示一个文本或图象。
Menu：菜单条。用来实现下拉和弹出式菜单。
Menubutton：菜单按钮。用来实现下拉式菜单。
Message：显示一文本。类似label窗口部件，但是能够自动地调整文本到给定的宽度或比率。
Text：格式化文本显示。允许你用不同的样式和属性来显示和编辑文本。同时支持内嵌图象和窗口。

In [11]:

```
wraptype = "char"
tabSpace = 4
openFiles = []
selectedFiles = 0
themeColors = []

class TextLineNumbers(tk.Canvas): #获取代码行数

    def __init__(self, *args, **kwargs):
        tk.Canvas.__init__(self, *args, **kwargs)
        self.textwidget = None

    def attach(self, text_widget):
        self.textwidget = text_widget

    def redraw(self, *args):
        self.delete("all")
        i = self.textwidget.index("@0,0")
        while True:
            dline = self.textwidget.dlineinfo(i)
            if dline is None:
                break
            y = dline[1]
            linenum = str(i).split(".")[0]
            self.create_text(5, y, anchor="nw", text=linenum, font=("Arial", 11))
            i = self.textwidget.index("%s+1line" % i)

class CustomText(tk.Text):
    def __init__(self, *args, **kwargs):
        tk.Text.__init__(self, *args, **kwargs)
        self.tk.eval('''
            proc widget_proxy {widget widget_command args} {
                # call the real tk widget command with the real args
                set result [uplevel [linsert $args 0 $widget_command]]
                # generate the event for certain types of commands
                if {([lindex $args 0] in {insert replace delete}) ||
                    ([lrange $args 0 2] == {mark set insert}) ||
                    ([lrange $args 0 1] == {xview moveto}) ||
                    ([lrange $args 0 1] == {xview scroll}) ||
                    ([lrange $args 0 1] == {yview moveto}) ||
                    ([lrange $args 0 1] == {yview scroll})} {
                    event generate  $widget <<Change>> -when tail
                }
                # return the result from the real widget command
                return $result
            }
            ''')
        self.tk.eval('''
            rename {widget} _{widget}
            interp alias {{}} ::{widget} {{}} widget_proxy {widget} _{widget}
        '''.format(widget=str(self)))
        self.comment = False
        self.bind("<Tab>", self.indent)

    def indent(self, arg):   #实现缩进
        self.insert(tk.INSERT, " " * tabSpace)
        return 'break'

    def copy(self):   #实现复制
```

```
        self.clipboard_clear()
        text = self.get("sel.first", "sel.last")
        self.clipboard_append(text)

    def configureBackground(self, background):   #设置背景
        self.configure(bg=background)

class Tab:
    def __init__(self, parent, filename, parentwindow): #初始化window
        self.content = ""
        self.previousContent = ""
        self.parentwindow = parentwindow
        tabNoBorder = ttk.Style()
        tabNoBorder.layout("Tab",
                           [('Notebook.tab', {'sticky': 'nswe', 'children':
                               [('Notebook.padding', {'side': 'top', 'sticky': 'nswe', 'children':
                                   [('Notebook.label', {
                                       'side': 'top', 'sticky': ''})],
                                                    })],
                                           })]
                           )
        self.parent = parent
        self.filename = filename
        self.tab1 = ttk.Frame(parent, style="Tab")
        try:
            if wraptype == "word":
                self.text = CustomText(self.tab1, bd=0, font=("Lucida Console", 11), undo=True,
                                       background=themeColors[0].strip(), foreground=themeColors[1].
                                       insertbackground=themeColors[2].strip(), wrap=tk.WORD)
            else:
                self.text = CustomText(self.tab1, bd=0, font=("Lucida Console", 11), undo=True,
                                       background=themeColors[0].strip(), foreground=themeColors[1].
                                       insertbackground=themeColors[2].strip(), wrap=tk.CHAR)
        except:
            if wraptype == "word":
                self.text = CustomText(self.tab1, bd=0, font=(
                    "Lucida Console", 11), undo=True, background="#454545", foreground="#FAFAFA",
                                       insertbackground="#FAFAFA", wrap=tk.WORD)
            else:
                self.text = CustomText(self.tab1, bd=0, font=(
                    "Lucida Console", 11), undo=True, background="#454545", foreground="#FAFAFA",
                                       insertbackground="#FAFAFA", wrap=tk.CHAR)
        self.row = "0"
        self.column = "0"
        self.startCol = 0
        self.vsb = ttk.Scrollbar(self.tab1, orient=tk.VERTICAL)
        self.text.configure(yscrollcommand=self.vsb.set)
        self.vsb.configure(command=self.text.yview)
        self.linenumbers = TextLineNumbers(self.tab1, width=32)
        self.linenumbers.attach(self.text)
        self.vsb.pack(side=tk.RIGHT, fill=tk.Y)
        self.linenumbers.pack(side="left", fill="y")
        self.text.pack(side="right", fill="both", expand=True)
        fileparts = filename.split("/")
        parent.add(self.tab1, text=fileparts[len(fileparts) - 1])
        self.fileOpened = "Untitled Document"
        try:
            if "Untitled Document" != filename:
                self.fileOpened = filename
                contentStuff = ""
                with open(filename, 'r') as file:
```

```
                for i in file.readlines():
                    contentStuff += i
            self.text.insert("0.0", contentStuff)
            self.highlight("arg")
    except:
        pass
    #事件绑定
    self.text.bind("<<Change>>", self._on_change)   #自定义事件
    self.text.bind("<Configure>", self._on_change)  #自定义事件
    self.text.bind("<KeyRelease>", self.keypress)   #键盘输入
    self.text.bind("<Button-1>", self.keypress)     #鼠标左键
    self.parent = parent
    self.configureTags()

def configureTags(self):   #根据token类型设置不同的颜色
    try:
        self.text.tag_configure("Token.Keyword", foreground=themeColors[3].strip())
        self.text.tag_configure("Token.Keyword.Constant", foreground=themeColors[4].strip())
        self.text.tag_configure("Token.Keyword.Declaration", foreground=themeColors[5].strip())
        self.text.tag_configure("Token.Keyword.Namespace", foreground=themeColors[6].strip())
        self.text.tag_configure("Token.Keyword.Pseudo", foreground=themeColors[7].strip())
        self.text.tag_configure("Token.Keyword.Reserved", foreground=themeColors[8].strip())
        self.text.tag_configure("Token.Keyword.Type", foreground=themeColors[9].strip())
        self.text.tag_configure("Token.Name.Class", foreground=themeColors[10].strip())
        self.text.tag_configure("Token.Name.Exception", foreground=themeColors[11].strip())
        self.text.tag_configure("Token.Name.Function", foreground=themeColors[12].strip())
        self.text.tag_configure("Token.Name.Tag", foreground=themeColors[13].strip())
        self.text.tag_configure("Token.Name.Builtin", foreground=themeColors[14].strip())
        self.text.tag_configure("Token.Operator.Word", foreground=themeColors[15].strip())
        self.text.tag_configure("Token.Comment", foreground=themeColors[16].strip())
        self.text.tag_configure("Token.Literal.String", foreground=themeColors[17].strip())
        self.text.tag_configure("Token.Literal.Number.Integer", foreground=themeColors[18].strip
        self.text.tag_configure("Token.Literal.Number.Bin", foreground=themeColors[19].strip())
        self.text.tag_configure("Token.Literal.Number.Float", foreground=themeColors[20].strip()
        self.text.tag_configure("Token.Literal.Number.Hex", foreground=themeColors[21].strip())
        self.text.tag_configure("Token.Literal.Number.Integer.Long", foreground=themeColors[22].
        self.text.tag_configure("Token.Literal.Number.Oct", foreground=themeColors[23].strip())
    except:
        self.text.tag_configure("Token.Keyword", foreground="#69A2DB")
        self.text.tag_configure("Token.Keyword.Constant", foreground="#69A2DB")
        self.text.tag_configure("Token.Keyword.Declaration", foreground="#69A2DB")
        self.text.tag_configure("Token.Keyword.Namespace", foreground="#D771D7")
        self.text.tag_configure("Token.Keyword.Pseudo", foreground="#69A2DB")
        self.text.tag_configure("Token.Keyword.Reserved", foreground="#69A2DB")
        self.text.tag_configure("Token.Keyword.Type", foreground="#69A2DB")
        self.text.tag_configure("Token.Name.Class", foreground="#8686D6")
        self.text.tag_configure("Token.Name.Exception", foreground="#8686D6")
        self.text.tag_configure("Token.Name.Function", foreground="#85D6FF")
        self.text.tag_configure("Token.Name.Tag", foreground="#8686D6")
        self.text.tag_configure("Token.Name.Builtin", foreground="#8686D6")
        self.text.tag_configure("Token.Operator.Word", foreground="#29A6CF")
        self.text.tag_configure("Token.Comment", foreground="#FF8A8A")
        self.text.tag_configure("Token.Literal.String", foreground="#5CA65C")
        self.text.tag_configure("Token.Literal.Number.Integer", foreground="#FF7DBD")
        self.text.tag_configure("Token.Literal.Number.Bin", foreground="#ACC3F2")
        self.text.tag_configure("Token.Literal.Number.Float", foreground="#7DA1EB")
        self.text.tag_configure("Token.Literal.Number.Hex", foreground="#5C8AE6")
        self.text.tag_configure("Token.Literal.Number.Integer.Long", foreground="#7DA1EB")
        self.text.tag_configure("Token.Literal.Number.Oct", foreground="#5C8AE6")

def deafultHighlight(self, argument):
```

```
        self.content = self.text.get("1.0", tk.END)
        self.lines = self.content.split("\n")
        self.row = self.text.index(tk.INSERT).split(".")[0]
        self.column = self.text.index(tk.INSERT).split(".")[1]
        self.text.mark_set("range_start", self.row + ".0")
        data = self.text.get(self.row + ".0", self.row + "." + str(len(self.lines[int(self.row) - 1]
        tokens = ["Token.Keyword", "Token.Keyword.Constant", "Token.Keyword.Declaration", "Token.Key
                  "Token.Keyword.Pseudo",
                  "Token.Keyword.Reserved", "Token.Keyword.Type", "Token.Name.Class", "Token.Name.Ex
                  "Token.Name.Function",
                  "Token.Name.Tag", "Token.Name.Builtin", "Token.Operator.Word", "Token.Comment",
                  "Token.Literal.String", "Token.Literal.Number.Integer",
                  "Token.Literal.Number.Bin", "Token.Literal.Number.Float", "Token.Literal.Number.He
                  "Token.Literal.Number.Integer.Long", "Token.Literal.Number.Oct"]
        for token in tokens:
            self.text.tag_remove(token, self.row + ".0", self.row + "." + str(len(self.lines[int(sel
        for token, content in lex(data, PythonLexer()):
            self.text.mark_set("range_end", "range_start + %dc" % len(content))
            self.text.tag_add(str(token), "range_start", "range_end")
            self.text.mark_set("range_start", "range_end")

    def highlight(self, argument):
        self.content = self.text.get("1.0", tk.END)
        if (self.previousContent != self.content):
            self.text.mark_set("range_start", "1.0")
            data = self.text.get("1.0", self.text.index(tk.INSERT))
            for token, content in lex(data, PythonLexer()):
                self.text.mark_set("range_end", "range_start + %dc" % len(content))
                self.text.tag_add(str(token), "range_start", "range_end")
                self.text.mark_set("range_start", "range_end")
        self.previousContent = self.text.get("1.0", tk.END)

    def displayFile(self, text):
        self.text.delete(0.0, tk.END)
        self.text.insert(0.0, text)
        self.highlight("Positional Argument")

    def getContent(self):
        return self.text.get(0.0, tk.END)

    def keypress(self, argument):
        self.deafultHighlight("argument")
        self.parent._nametowidget(self.parent.winfo_parent()).updateBottomLabel(self.text.index(tk.I
        self.text.tag_delete("Error")

    def replace(self, content):
        self.text.delete(1.0, tk.END)
        self.text.insert(1.0, content)

    def _on_change(self, event):
        self.linenumbers.redraw()

    def configureFont(self, fontFamily, fontSize):
        self.text.config(font=(fontFamily, fontSize))

class sublime(tk.Frame):        #界面类
    def __init__(self, *args, **kwargs):   #初始化
        tk.Frame.__init__(self, *args, **kwargs)
        self.tabs = []
        self.notebook = ttk.Notebook(self)
        self.fileName = ""
```

```
        self.content = ""
        self.highlightColor = "#000000"
        self.bottomlabel()
        self.createtext()
        self.menubar()
        self.instance = 0
        #设置快捷键
        self.bind_all("<Control-n>", self.newFile)
        self.bind_all("<Control-o>", self.openFile)
        self.bind_all("<Control-s>", self.saveFile)
        self.bind_all("<Control-S>", self.saveAsFile)
        self.bind_all("<Control-q>", self.close)
        self.bind_all("<Control-t>", self.addtab)
        self.bind_all("<Control-w>", self.removetab)
        self.bind_all("<Control-e>", self.changeFont)
        self.bind_all("<Control-k>", self.nextTab)
        self.bind_all("<F11>", self.toggleScreenSize)
        self.bind_all("<F1>", self.jumpToTop)
        self.bind_all("<F2>", self.jumpToBottom)
        self.lineNumbers = False
        self.bottomLabel = False
        self.syntaxHighlighting = True
        self.previousColor = ""
        self.standardColor = "#66FF66"
        self.previousRange = -4463
        self.previousCaseOrNot = False
        self.previousRegex = False
        self.previousContent = ""
        self.language = "Plain Text"
        self.font = "Consolas"
        self.fontSize = "11"
        #设置鼠标右键事件
        self.contextMenu = tk.Menu(self, font=("Consolas", 9), tearoff=0)
        self.contextMenu.add_command(label="Undo", command=self.undo)
        self.contextMenu.add_command(label="Redo", command=self.redo)
        self.contextMenu.add_separator()
        self.contextMenu.add_command(label="Cut", command=self.cut)
        self.contextMenu.add_command(label="Copy", command=self.copy)
        self.contextMenu.add_command(label="Paste", command=self.paste)
        self.contextMenu.add_separator()
        self.contextMenu.add_command(label="Remove Tab", command=lambda: self.removetab("arg"))
        self.bind_all("<Button-3>", self.popup)
    def popup(self, event):  #激活事件
        self.contextMenu.post(event.x_root, event.y_root)

    def createtext(self):
        self.notebook.pack(fill=tk.BOTH, expand=True)
        t = Tab(self.notebook, "Untitled Document", self)
        self.tabs.append(t)

    def menubar(self):  #添加菜单
         # Menu：菜单条，用来实现下拉和弹出式菜单，点下菜单后弹出的一个选项列表,用户可以从中选择
        self.menu = tk.Menu(self)
        self.master.config(menu=self.menu)
        # 在File中加入New File、Open File、Save File等小菜单，即我们平时看到的下拉菜单，每一个小菜单
        self.fileMenu = tk.Menu(self.menu, font=("Consolas", 9), tearoff=0)
        self.fileMenu.add_command(label="New File                    Ctrl+N", command=lambda: self.newF
        self.fileMenu.add_command(label="Open File                   Ctrl+O", command=lambda: self.oper
        self.fileMenu.add_command(label="Save File                   Ctrl+S", command=lambda: self.save
        self.fileMenu.add_command(label="Save As File       Ctrl+Shift+S", command=lambda: self.save
        self.fileMenu.add_separator()# 添加一条分隔线
```

```
        self.fileMenu.add_command(label="New Tab                  Ctrl+T", command=lambda: self.addt
        self.fileMenu.add_command(label="Close Tab                Ctrl+W", command=lambda: self.remo
        self.fileMenu.add_separator()# 添加一条分隔线
        self.fileMenu.add_command(label="Next Tab                 Ctrl+K", command=lambda: self.next
        self.fileMenu.add_separator()# 添加一条分隔线
        self.fileMenu.add_command(label="Exit                     Ctrl+Q", command=lambda: self.clos
        self.menu.add_cascade(label="File", menu=self.fileMenu)# 给放入的菜单fileMenu命名为File
        #同理，为Edit加入Font、ind and Replace等小菜单
        self.editMenu = tk.Menu(self.menu, font=("Consolas", 9), tearoff=0)
        self.editMenu.add_command(label="Font                       Ctrl+E", command=lambda: self.ch
        self.editMenu.add_separator()
        self.editMenu.add_command(label="Copy                       Ctrl+C", command=self.copy)
        self.editMenu.add_command(label="Paste                      Ctrl+V", command=self.paste)
        self.editMenu.add_command(label="Cut                        Ctrl+X", command=self.cut)
        self.editMenu.add_separator()
        self.editMenu.add_command(label="Undo                       Ctrl+Z", command=self.undo)
        self.editMenu.add_command(label="Redo                       Ctrl+Y", command=self.redo)
        self.editMenu.add_separator()
        self.editMenu.add_command(label="Jump to top                    F1", command=lambda: self.ju
        self.editMenu.add_command(label="Jump to end                    F2", command=lambda: self.ju
        self.menu.add_cascade(label="Edit", menu=self.editMenu) # 给放入的菜单editMenu命名为Edit
        self.selectionMenu = tk.Menu(self.menu,font=("Consolas", 9), tearoff=0)
        self.menu.add_cascade(label="Selection", menu=self.selectionMenu)
        self.findMenu = tk.Menu(self.menu,font=("Consolas", 9), tearoff=0)
        self.menu.add_cascade(label="Find", menu=self.findMenu)
        self.viewMenu = tk.Menu(self.menu, font=("Consolas", 9), tearoff=0)
        self.menu.add_cascade(label="View", menu=self.viewMenu)
        self.gotoMenu = tk.Menu(self.menu, font=("Consolas", 9), tearoff=0)
        self.menu.add_cascade(label="Goto", menu=self.gotoMenu)
        self.toolsMenu = tk.Menu(self.menu, font=("Consolas", 9), tearoff=0)
        self.menu.add_cascade(label="Tools", menu=self.toolsMenu)
        self.projectMenu = tk.Menu(self.menu, font=("Consolas", 9), tearoff=0)
        self.menu.add_cascade(label="Project", menu=self.projectMenu)
        self.preferencesMenu = tk.Menu(self.menu, font=("Consolas", 9), tearoff=0)
        self.menu.add_cascade(label="Preferences", menu=self.preferencesMenu)
        self.helpMenu = tk.Menu(self.menu, font=("Consolas", 9), tearoff=0)
        self.menu.add_cascade(label="Help", menu=self.helpMenu)
    def jumpToTop(self, arg):
        tabIndex = self.notebook.index(self.notebook.select())
        self.tabs[tabIndex].text.see("0.0")
        self.tabs[tabIndex]._on_change("arg")

    def jumpToBottom(self, arg):
        tabIndex = self.notebook.index(self.notebook.select())
        self.tabs[tabIndex].text.see(tk.END)
        self.tabs[tabIndex]._on_change("arg")

    def openFolder(self, arg):
        pass

    def undo(self):
        tabIndex = self.notebook.index(self.notebook.select())
        self.tabs[tabIndex].text.edit_undo()

    def redo(self):
        tabIndex = self.notebook.index(self.notebook.select())
        self.tabs[tabIndex].text.edit_redo()

    def cut(self):
        tabIndex = self.notebook.index(self.notebook.select())
        self.tabs[tabIndex].text.copy()
```

```
        self.tabs[tabIndex].text.delete("sel.first", "sel.last")

    def copy(self):
        tabIndex = self.notebook.index(self.notebook.select())
        self.tabs[tabIndex].text.clipboard_clear()
        text = self.tabs[tabIndex].text.get("sel.first", "sel.last")
        self.tabs[tabIndex].text.clipboard_append(text)

    def paste(self):
        tabIndex = self.notebook.index(self.notebook.select())
        text = self.tabs[tabIndex].text.selection_get(selection='CLIPBOARD')
        self.tabs[tabIndex].text.insert('insert', text)

    def toggleScreenSize(self, arg):
        self.fullScreen = not self.fullScreen
        self.master.attributes("-fullscreen", self.fullScreen)

    def changeFont(self, arg):
        self.fontOption = tk.Toplevel()
        self.fontOption.resizable(0, 0)
        self.fontOption.title("Choose Font")
        self.selectFont = tk.Label(self.fontOption, text="Font Family: ", font=("Consolas", 9))
        self.selectFont.grid(row=0, columnspan=1)
        self.fontComboBox = ttk.Combobox(self.fontOption)
        self.fontComboBox['values'] = ("Arial", "Courier New", "Consolas", "Georgia", "Monaco", "MS
                    "Lucida Grande", "Lucida Sans Unicode", "Tahoma", "Trebuchet MS", "Times New Rom
        self.fontComboBox.current(2)
        self.fontComboBox.grid(row=0, column=1, columnspan=1)
        self.selectFontSize = tk.Label(self.fontOption, text="Font Size: ", font=("Consolas", 9))
        self.selectFontSize.grid(row=1, columnspan=1)
        self.fontSizeComboBox = ttk.Combobox(self.fontOption)
        self.fontSizeComboBox['values'] = ("8", "9", "10", "11", "12", "13", "14", "15", "16", "17",
        self.fontSizeComboBox.current(3)
        self.fontSizeComboBox.grid(row=1, column=1, columnspan=1)
        self.fontProceed = tk.Button(self.fontOption, text="Ok", font=("Consolas", 9), command=self.
        self.fontProceed.grid(row=2, column=0, columnspan=2, sticky='NSEW')

    def proceedWithFontChange(self):
        self.fontFamily = self.fontComboBox.get()
        self.fontSize = self.fontSizeComboBox.get()
        if self.fontFamily not in ["Arial", "Courier New", "Consolas", "Georgia", "Monaco", "MS Sans
                                   "Lucida Grande", "Lucida Sans Unicode", "Tahoma", "Trebuchet MS",
            tkinter.messagebox.showinfo("Invalid Font Family", "sublime supports the current font op
                                        + "MS Serif, New York, Lucida Console, Lucida Grande, Lucida
            self.fontOption.destroy()
        elif self.fontSize not in ["8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "
            tkinter.messagebox.showinfo("Invalid Font Size", "sublime supports font sizes from 8px t
        else:
            tabIndex = self.notebook.index("end")
            for i in range(0, tabIndex):
                self.tabs[i].configureFont(self.fontFamily, int(self.fontSize))

    def nextTab(self, arg):
        tabIndex = self.notebook.index(self.notebook.select())
        tabIndex += 1
        if tabIndex == self.notebook.index("end"):
            tabIndex = 0
        self.notebook.select(tabIndex)

    def newFile(self, arg):
        tabIndex = self.notebook.index(self.notebook.select())
```

```
        self.notebook.forget(tabIndex)
        t = Tab(self.notebook, "Untitled Document", self)
        self.tabs.append(t)
        self.file = "Untitled Document"+str(self.tabs.__len__()-1)
        self.notebook.tab(tabIndex, text=self.file)
        self.notebook.select(tabIndex)

    def openFile(self, arg):
        try:
            self.fileName = tk.filedialog.askopenfilename()
            with open(self.fileName, 'r') as file:
                self.content = file.read()
            tabIndex = self.notebook.index(self.notebook.select())
            self.tabs[tabIndex].displayFile(self.content)
            locations = self.fileName.split("/")
            self.file = locations[len(locations) - 1]
            self.notebook.tab(tabIndex, text=self.file)
            self.tabs[tabIndex].fileOpened = self.fileName
        except IOError as e:
            pass

    def deleteContent(self, file):
        file.seek(0)
        file.truncate()

    def saveFile(self, arg):
        tabindex = self.notebook.index(self.notebook.select())
        self.content = self.tabs[tabindex].getContent()
        try:
            with open(self.tabs[tabindex].fileOpened, 'w') as file:
                self.deleteContent(file)
                file.write(self.content)
        except IOError as e:
            pass
        except:
            pass

    def saveAsFile(self, arg):
        tabIndex = self.notebook.index(self.notebook.select())
        self.content = self.tabs[tabIndex].getContent()
        try:
            self.fileName = tk.filedialog.asksaveasfilename()
            if self.fileName != None:
                with open(self.fileName, 'w') as file:
                    file.write(self.content)
            locations = self.fileName.split("/")
            self.file = locations[len(locations) - 1]
            self.notebook.tab(tabIndex, text=self.file)
            self.tabs[tabIndex].fileOpened = self.fileName
        except IOError as e:
            pass
        except:
            pass

    def addtab(self, arg):
        t = Tab(self.notebook, "Untitled Document", self)
        self.tabs.append(t)
        self.notebook.select(self.notebook.index("end") - 1)
        self.tabs[self.notebook.index(self.notebook.select())].text.focus_set()
        self.tabs[self.notebook.index(self.notebook.select())].text.configure(font=(self.font, self.
```

```
    def bottomlabel(self):
        self.positionAndLanguage = tk.Label(
            self, text=" Ln: 1, Col: 0", anchor=tk.W, bg="#E7E7E7", font=("Arial", 9))
        self.positionAndLanguage.pack(fill=tk.X, side=tk.BOTTOM)

    def updateBottomLabel(self, line, column, length, language):
        self.positionAndLanguage["text"] = " Ln: {0}, Col: {1}, Length: {2}".format(line, column, st

    def removetab(self, arg):
        numberOfTabs = self.notebook.index("end")
        if numberOfTabs > 1:
            tabIndex = self.notebook.index(self.notebook.select())
            self.notebook.forget(tabIndex)
            self.tabs[self.notebook.index(self.notebook.select())].text.focus_set()
            del self.tabs[tabIndex]

    def protocol(self, arg2, arg3):
        self.master.protocol(arg2, arg3)

    def close(self, arg):
        try:
            os._exit(0)
        except:
            pass
def mainCloseProtocol(root, window, wraptype):
    root.destroy()
```

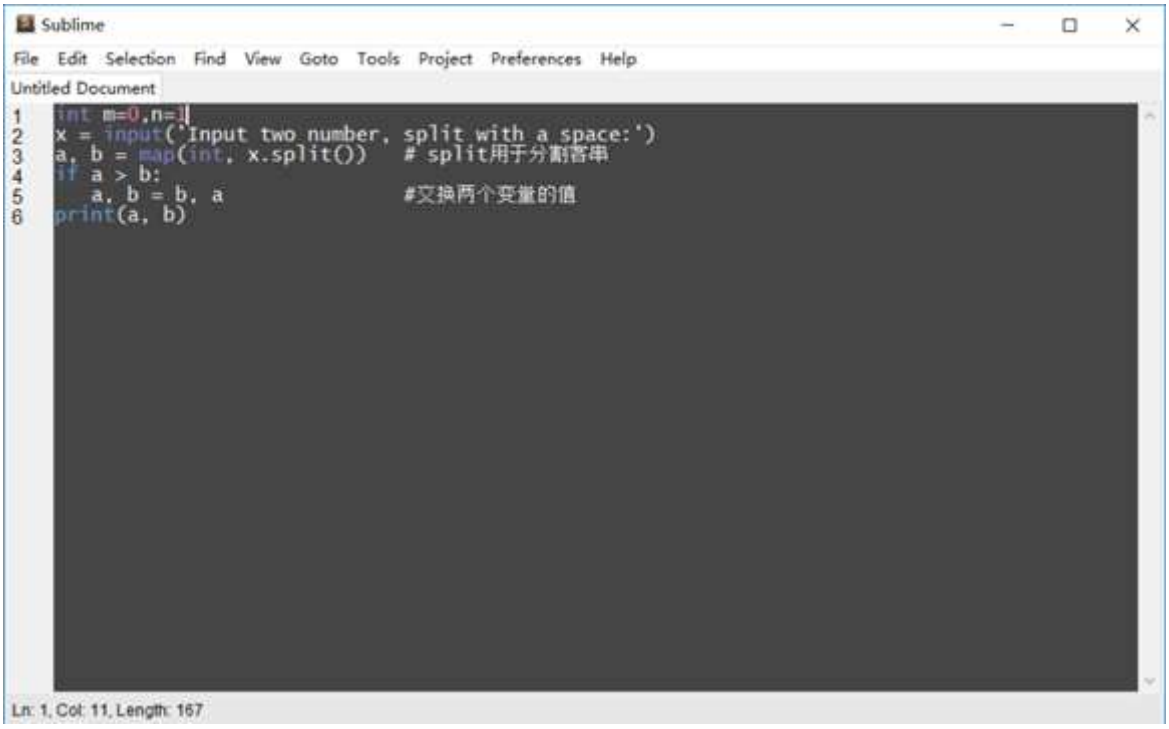
In [12]:

```
if __name__ == "__main__":
    root = tk.Tk() #实例化object，建立窗口window
    img = tk.PhotoImage(file='icon.PNG') # 说明图片位置，并导入图片到画布上
    root.tk.call('wm', 'iconphoto', root._w, img)
    root.title("Sublime") # 给窗口的可视化起名字
    root.geometry("1024x600") #设定窗口的大小(长 * 宽)
    window = sublime(root)
    window.pack(side="top", fill="both", expand=True) #放置方法
    window.protocol("WM_DELETE_WINDOW", lambda: mainCloseProtocol(root, window, wraptype))
    root.mainloop() # 主窗口循环显示
```

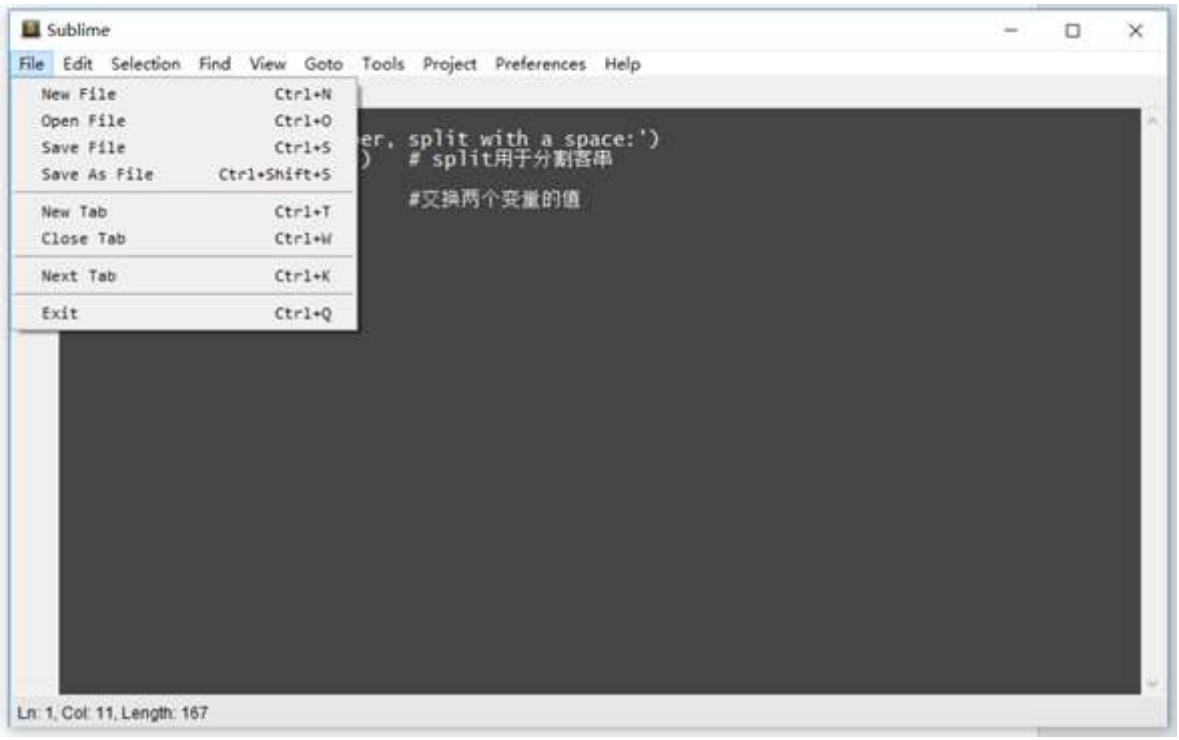
注意，loop因为是循环的意思，window.mainloop就会让window不断的刷新，如果没有mainloop,就是一个静态的window,传入进去的值就不会有循环，mainloop就相当于一个很大的while循环，有个while，每点击一次就会更新一次，所以我们必须要有循环所有的窗口文件都必须有类似的mainloop函数，mainloop是窗口文件的关键的关键。

运行截图：

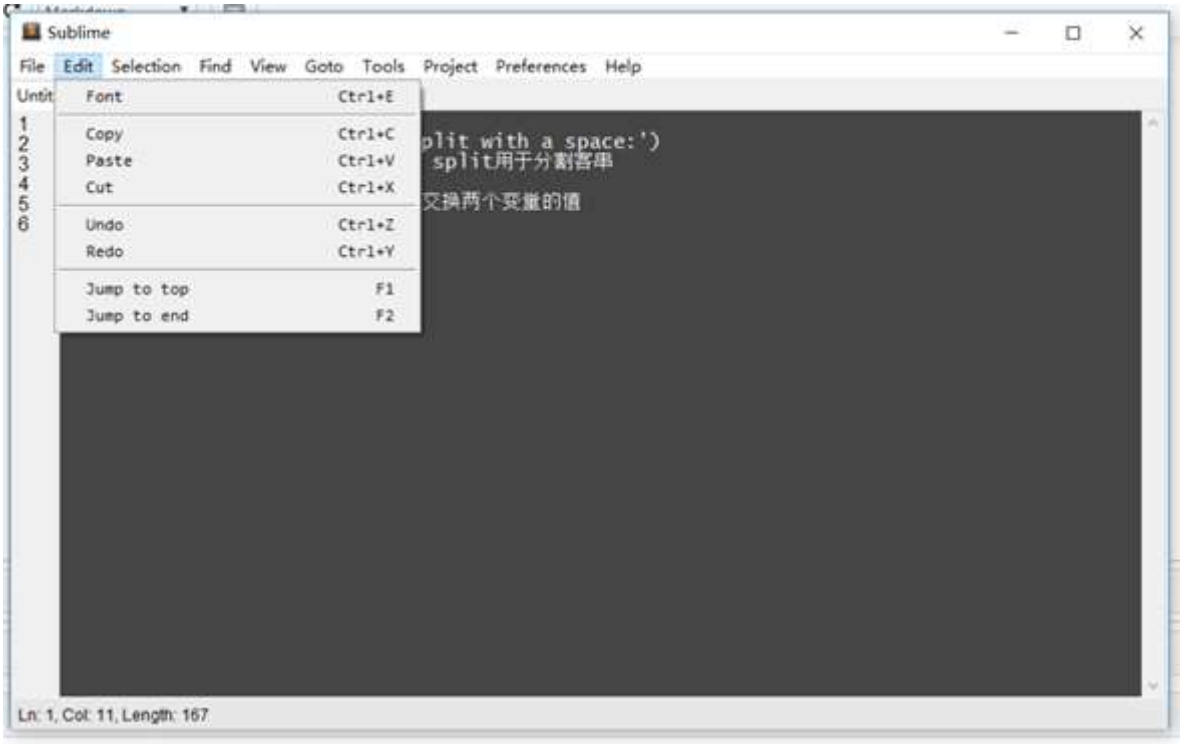
实现代码高亮：



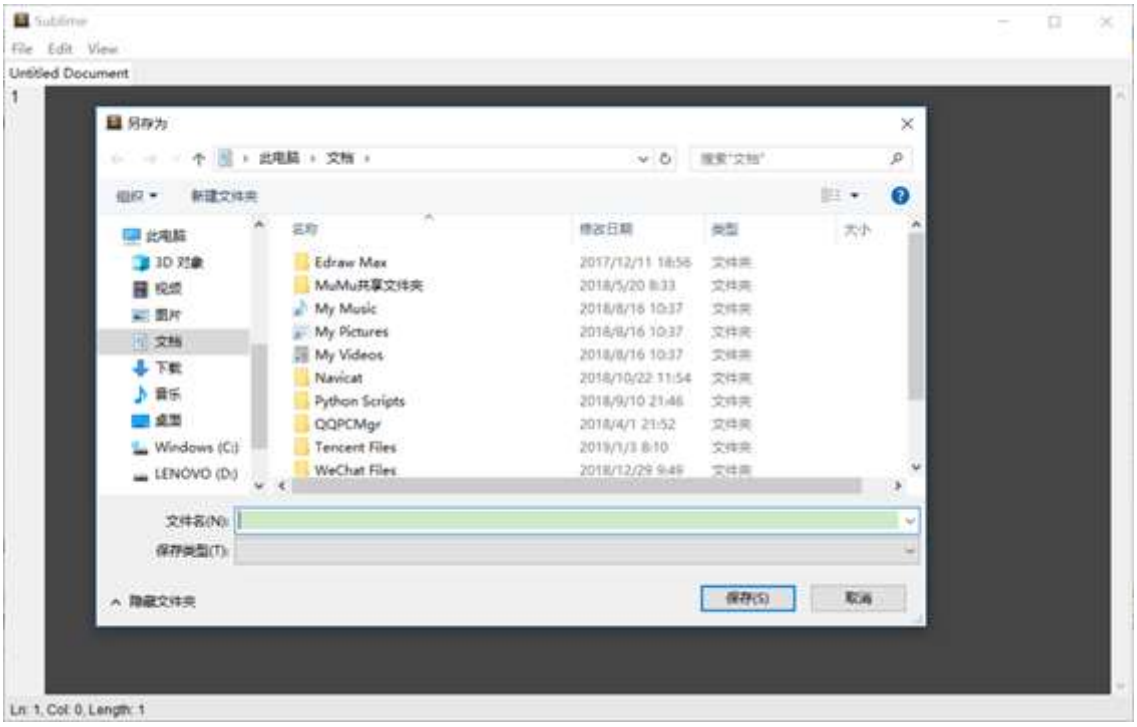
File菜单项功能：



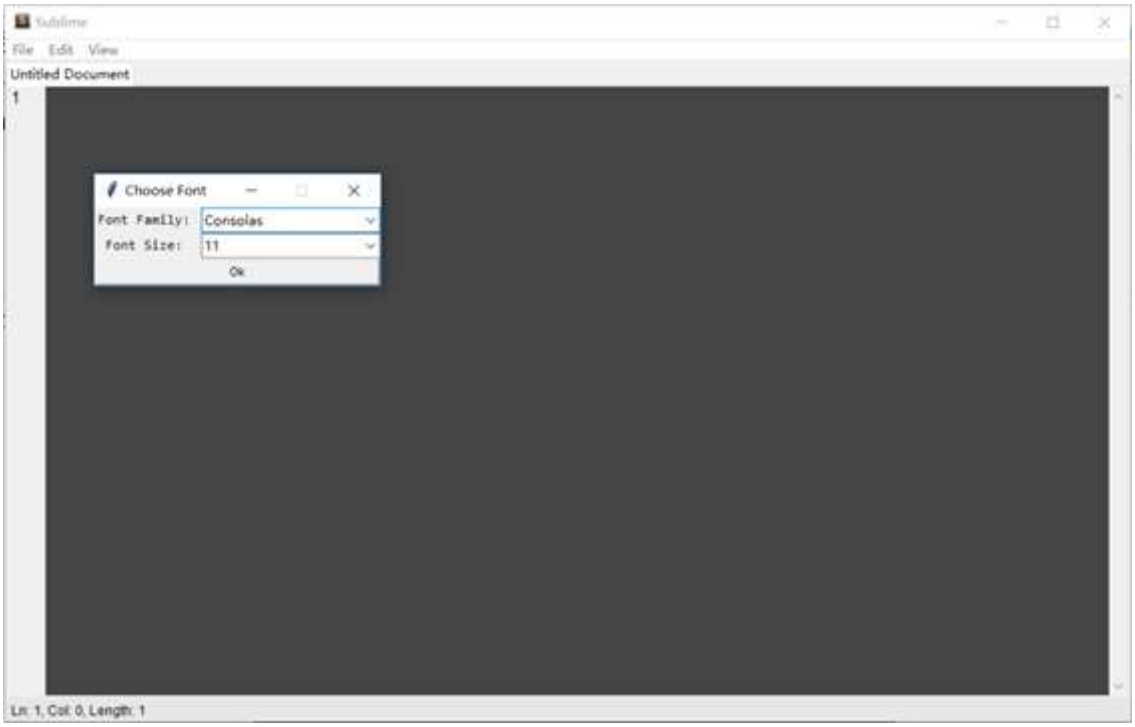
Edit菜单项功能：



保存文件，另存文件，打开文件的界面:



修改字体：



自我评价：

本次完成了一个简洁、易用的文本编辑器，可以用来编写python代码，实现了语法高亮功能，具有新建文件、打开文件、复制、粘贴、撤销、跳转顶部或底部等编辑器基础功能且具有对应快捷键，不足之处在于不能支持多语言，没有实现自动补足功能，自动提示语法和已定义过的变量或函数等高级要求没有实现。其实，抢到这道题也才近一周的时间，16周还有各种考试和课设，所以完成的不好的地方还请老师见谅。