Notepad using Tkinter with python- Tutorial

version 0.1 January 22, 2018

About

This is a tutorial where you will be using the **tkinter** module, which is an open source module available in the python official website, to create a gui application. You will get to know about the features and usage of the module tkinter in this tutorial. We recommend you to use an ide called **pycharm**(it is a pretty good ide for python, check it out) to run your code. This application will be a console application, do not worry, you will get to know what that means, if you are not familiar with the idea. You will also learn to make an executable file of our python code, which will be covered in the last section.

The operating system we are using in this tutorial is windows 7 64-bit, so some of the commands and procedures may be different if you are using a different os.

Table of contents

.

1. Introduction to GUI programming in Python

1.1. What is a GUI?

The **graphical user interface** (**GUI**), is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation. GUIs were introduced in reaction to the perceived steep learning curve of command-line interfaces (CLIs), which require commands to be typed on a computer keyboard.

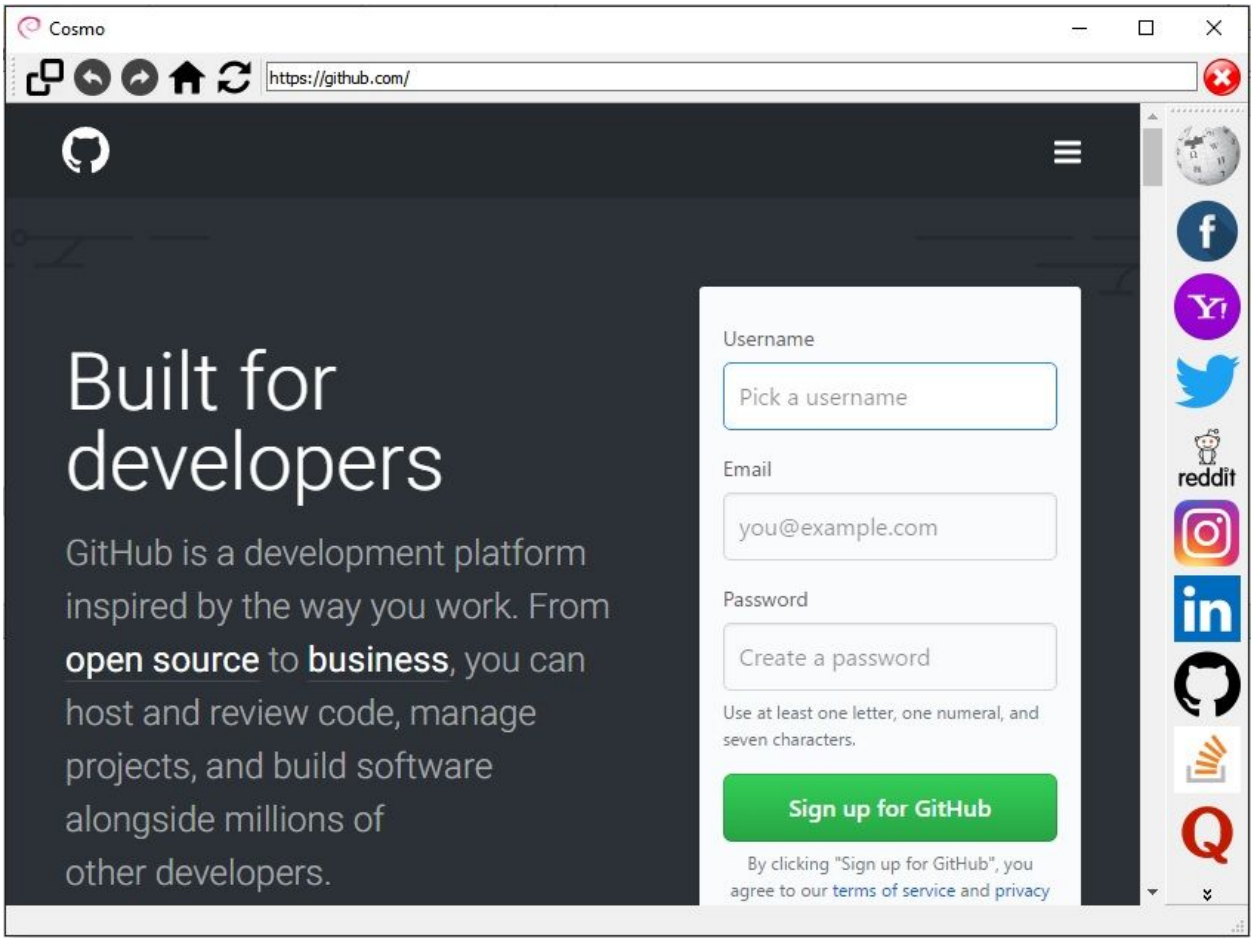
The actions in a GUI are usually performed through direct manipulation of the graphical elements. Beyond computers, GUIs are used in many handheld mobile devices such as MP3 players, portable media players, gaming devices, smartphones and smaller household, office and industrial controls.

1.2. GUI programming in Python

Python has a huge number of **GUI** frameworks (or toolkits) available for it, from TkInter (traditionally bundled with **Python**, using Tk) to a number of other cross-platform solutions, as well as bindings to platform-specific (also known as "native") technologies.

Apart from Tkinter Python toolkits include:

- PyQt
- PySide
- wxPython
- PyGTK, etc.



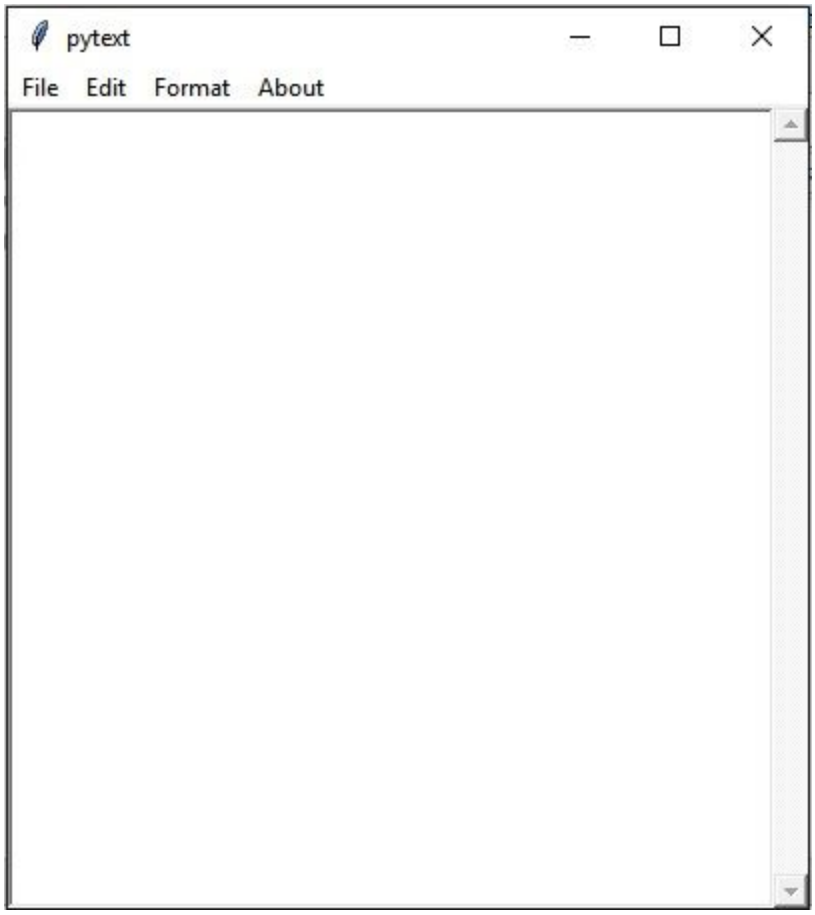
A browser built using pyqt

2. Our Notepad

In this project you will be making a notepad in Python using the module Tkinter. Here you will use the concepts of object oriented program and create a class named notepad and then create an object of the class notepad, named noteobj.

2.1 How will the final Application look like?

This is how the application will look like.



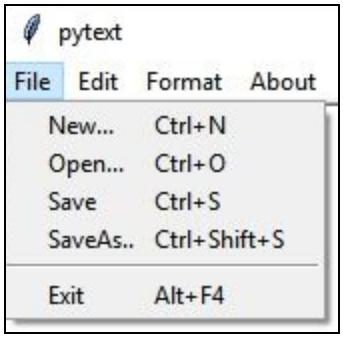
The notepad will have the following components:

- It will have a menu bar, which will contain the menus file menu, edit menu, format menu and About.
- Most obviously, it will have an area or space where you can type and edit any text.
- And a scroll bar

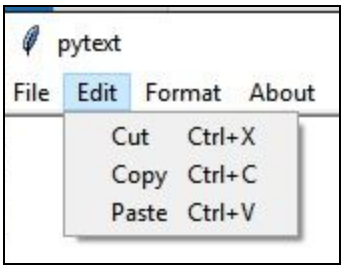
That's all for this project.

2.2 What will be the features of the application?

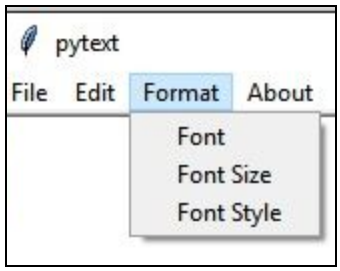
- The file menu will contain the options: **new file, open file, save file, saveAs file and exit.**



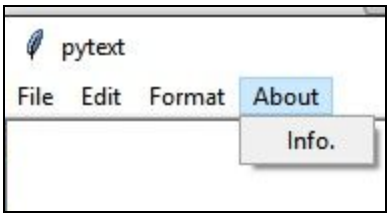
- The edit menu will contain the options: **cut, copy paste.**



- The format menu will contain the options: **font,font style, font size.**



- And the About menu will have the option **Info.**.



*We will try to add more features to our notepad in the near future.

3. Prerequisites

- You should be familiar with basic python syntax.
- You should be familiar with classes and objects in python.
- Your system should have python **3.4** installed(other python versions may not be handy with this tutorial's code).

4. Installing the tkinter module

Usually python comes with the tkinter module pre installed(check before going through the following overheads, just type import tkinter in your ide. If the ide does not show any error then tkinter is already installed), but if your system does not contain the module, dont worry, you can install this module by using a number of methods. We will show you one of the methods. Here we will be using the pip installer which is provided by python itself. The pip installer is located in python\scripts folder. Following these steps :

I. Open cmd if you are using windows(mac:bash; linux: terminal)[we will showing command for windows, if you are not using windows just search for terminal commands for your os)
II. Navigate to python-> scripts folder. You can do this with by typing **cd** <path> , for example **cd C:\Python34\Scripts.**
III. Then type pip install tkinter. The module will be installed automatically. You are done.

You can also set the path variable before installing the module. By setting the path variable you can use the pip or any other file in the scripts folder from any directory. Follow these steps :

I. Copy the address of the **pip.exe** file (it should be located in python->**scripts** folder).

II. Right click the **my computer**(or This PC) icon on your desktop. From the menu, select the **properties** option.
III. The system window will appear. You will find an option called **advanced system settings**, click it.
IV. The system properties dialog box will appear. In the **Advanced** tab select the **Environment variables...** option.
V. The Environment variables dialog box will appear. In the system variables area click on **path** and select **Edit.**
VI. The edit environment variable dialog box will appear. Select the **new** button and paste the address of the pip.exe file in the given space.
VII. Now just click ok one by one until all the dialog boxes are close. The path variable set and you are ready to install the tkinter module from any directory using cmd.

5. First steps to the code

5.1 Modules to import

```
from tkinter import *
import tkinter.scrolledtext as Text1      #Text1 is an alias
from tkinter.filedialog import *
from tkinter.messagebox import *
```

'*' is used to import all the submodules of the respective module

The module **tkinter.scrolledtext** enables you to scroll the the text space.

The module **tkinter.filedialog** provides us an easy way to create some of the important dialog boxes like save file, new file, etc.

The **tkinter.messagebox** provides api for message boxes which are used to create our notepad.

5.2 Defining the class

After importing the modules it's time to define the class notepad. This is how you should do it

```
class notepad:
```

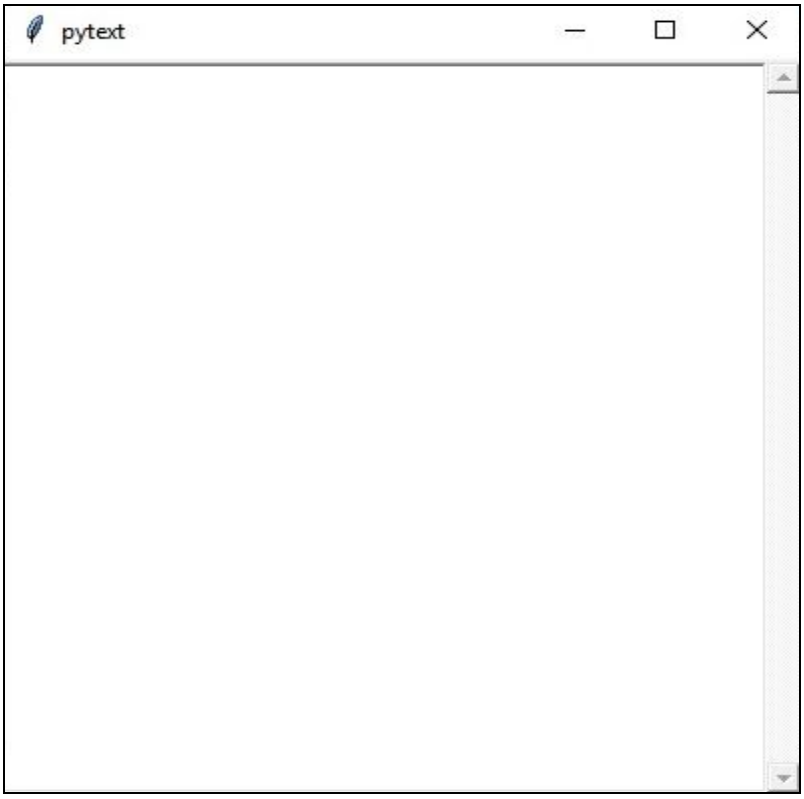
Under this class the whole code is written. Now let us look at the functions defined within this class and what are they used for.

6. Constructor of our class

The constructor initializes an object of our 'notepad' class at the moment the object is created. Following is the constructor function itself and is divided into three sections to increase understandability.

6.1 __init__(part 1)

```
def __init__(self):
    #CONSTRUCTOR FUNCTION
    self.root=Tk()                          #Creates a window root of the class Tk()
    self.root.title("pytext")               #sets the title of the window
    self.root.geometry("400x400")           #sets the default size of the window
    self.fontstyle="normal"
    self.font="Courier New"
    self.fontsize=12
    self.text=Text1.ScrolledText(self.root,relief="sunken",bd=2,width=400,height=400)
    #Creates a text space which can be scrolled(relief gives a custom boundary)
    self.text.pack()                        # packs the text space in root
    self.firstsave=0                        # variable to store save status of a file
    self.fontvar = StringVar()              # font variable for radiobutton.
    self.fontvar.set(self.font)             # sets self.fontvar to self.font
    self.style = StringVar()                # font style variable for radiobutton
    self.style.set(self.fontstyle)
    self.text.config(font=(self.font, self.fontsize, self.fontstyle,))
    # Set the font, font size and font style:
```



You will understand the use of the variables initialized above as the tutorial progresses.

When you run the above code a window like this should appear. (self.root.mainloop(), add this line to the above code to execute itt properly)

6.2 __init__(part 2)

This snippet contains the coding part used to create your notepad's menu bar.

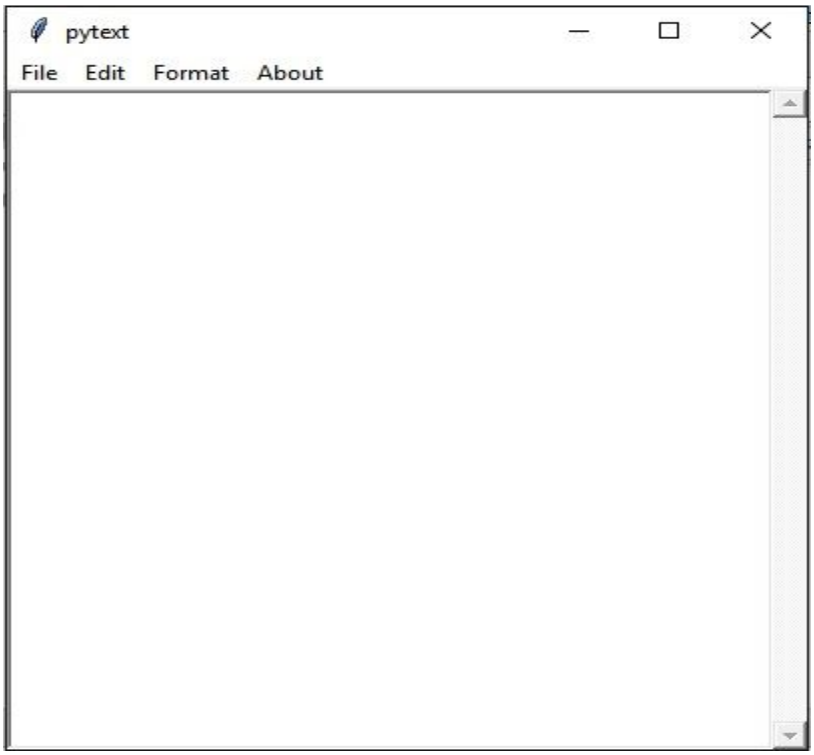
```
# MENU BAR
menu = Menu(self.root)                  #creates a menubar named 'menu' under root
self.root.config(menu=menu)             #adds the menu bar to our program

filemenu = Menu(menu,tearoff=0)                  #creates a menu in the menu bar; exp2
menu.add_cascade(label="File", menu=filemenu)   #adds the option file to the menu bar

filemenu.add_command(label="New...", command=self.newFile, accelerator="Ctrl+N")# exp1
filemenu.add_command(label="Open...", command=self.openFile,accelerator="Ctrl+O")
filemenu.add_command(label="Save", command=self.saveFile,accelerator="Ctrl+S")
filemenu.add_command(label="SaveAs..", command=self.saveAs,accelerator="Ctrl+Shift+S")
filemenu.add_separator()                #adds a separator in the submenu list
filemenu.add_command(label="Exit", command=self.Exit,accelerator="Alt+F4")
```

Exp1: tearoff is used clear off the unwanted '----------' option in each submenu, write the code without the tearoff attribute and you will be able to figure the difference.

Exp2: Creates the options in the submenu of file menu.



You can add the other menus and submenus similarly. The entire code is provided in the resources section.

After adding all the menus and submenus your application will look like this.

6.3 __init__(part3)

This snippet contains the coding part used to add some keyboard shortcuts in your application.

```
#keyboard shortcuts
self.root.bind("<Control-N>",self.newFile)
self.root.bind("<Control-O>",self.openFile)
self.root.bind("<Control-o>",self.openFile)
```

Similarly you can add the other keyboard shortcuts. We also need to add an **event** to each binded function which will be discussed in section 7.2 Newfile(Note).

At the end of the constructor function write the following line which keeps our application on the screen and our application is not terminated automatically.

```
self.root.mainloop()          # keeps the window on screen
```

7. File menu

This section will deal with the creation of all the option of the file menu.

7.2 New file

```
def newFile(self,event=NONE):
    if self.firstsave==0:
        self.newsave()
    else:
        self.filename = "Untitled"
        self.text.delete(0.0, END)
        self.firstsave=0
```

If the file is not saved then the newsave function will be called. Else the text in the text space will be deleted, the file name will be changed to default name and the save status will be set to 0.

Note: *The event attribute in the definition of the function specifies that this function can be invoked when a specified event undertakes(for example: mouse click, key presses).*

```
def newsave(self):
    f = asksaveasfile(title="Save current file!",
      defaultextension = ".txt" , filetypes=[("text file", "*.txt")])
    t = self.text.get(0.0, END)
    try:
```

```
            f.write(t.rstrip())
            self.firstsave = 1
            self.newFile()
        except:
            pass
```

When invoked this function opens the save dialog box. If the save operation is successful it calls the newfile function. Else the function control passes to newfile function without saving the file.

7.3 Open file

```
def openFile(self,event=NONE):
    try:
        f=askopenfile(mode='r', filetypes=[("text file", "*.txt")])
                            #displays a dialog box to open a saved
file
        self.filename=f.name      #gets the file name of the opened
file
                                  and makes the current file name
        t=f.read()                #reads the text in the file
        self.text.delete(0.0,END) #deletes the previous text
        self.text.insert(0.0,t)   #inserts new text from the opened
file
        self.firstsave=1           #changes the save status to 1
    except:
        Pass
```

When the open option is clicked under the file menu this function is called. All the lines are explained as you can see.

7.4 Save file

This function is almost similar to the last two functions. Small changes are explained in the commented parts.

```
def saveFile(self,event=NONE):
   if self.firstsave==0:
       f = asksaveasfile(title="Save",defaultextension=".txt", filetypes=[("text
                         file", "*.txt")])     #opens the save dialog box
       t = self.text.get(0.0, END)
       try:
           f.write(t.rstrip())#writes the text present in the text space in the file f
           self.firstsave = 1
       except:
           pass
   Else:                                  # if the file is already saved, no dialog
       t = self.text.get(0.0, END)          box is opened and the file is saved directly
       f = open(self.filename, 'w')
       f.write(t)
       f.close()
       self.firstsave = 1
```

7.5 SaveAs file

```
def saveAs(self,event=NONE):
   f = asksaveasfile(title="SaveAs",defaultextension=".txt", filetypes=[("text
file", "*.txt")])
   t = self.text.get(0.0, END)
   try:
       f.write(t.rstrip())
       self.firstsave = 1
   except:
       pass
```

It is expected that you will be able understand this function. It is almost similar to the previous function.

7.6 Exit file

```
def Exit(self,event=NONE):
   if self.firstsave==0 and self.text.get(0.0,END)!="\n":
       Userinput = askquestion("File not saved.","Do you want to save
```

```
                                    this file?")
        if userinput=='yes':
            self.saveAs()
    self.root.quit()
```

This function is used to exit the application. If the current file is not saved a message box appears asking the user whether he/she wants to save the file or not.
Lastly it calls the quit() function of class root to exit the application.

8. Edit menu

This section will deal with the creation of all the option of the edit menu. Before getting to the functions you need to know what is clipboard. The clipboard stores data temporarily that allows both read and write operations. The clipboard will be used in the editing functions.

8.1 Copy text

```
def Copy(self,event=NONE):
    self.root.clipboard_clear()                    #clipboard is cleared
    self.text.clipboard_append(string=self.text.selection_get())
    #the selected text is appended in the empty clipboard
```

8.2 Cut text

```
def Cut(self,event=NONE):
    self.root.clipboard_clear()
    self.text.clipboard_append(string=self.text.selection_get())
    self.text.delete(index1=SEL_FIRST,index2=SEL_LAST)
    #the selected text is deleted (indexes are used to specify the
     first and last selected character).
```

8.3 Paste text

```
def Paste(self,event=NONE):
    self.text.insert(INSERT,self.root.clipboard_get())
    #inserts the text present in the clipboard at the current cursor
     position)
```

9. Format menu

This section will deal with the creation of all the option of the format menu.
Before typing the formatting functions, let us discuss radio buttons.
A **radio button**, sometimes called **option button**, is a graphical user interface element of **Tkinter**, which allows the user to choose (exactly) one of a predefined set of options. **Radio buttons** can contain text or images. The**button** can only display text in a single font.
It has the following attributes:

- Text: Displayed text for each radio button
- Value: Stores a value which is different for each radio button
- Variable: A variable that stores the content of 'value' of the current selected radio button

9.1 Changing font

This section consists of the functions needed to change the font.

9.1.1 Main function

```
def Font(self):
    self.top = Toplevel()   #Toplevel is a class used to create small pop-up windows
    self.top.title("Font")

    label = Label(self.top, text="Please select a font...", width=30)
    label.pack()
     #creates a label and packs it, which will appear on the window
    fonts = (
```

```
        "Arial",
        "Courier New",              #creates a tuple containing different fonts
        "Verdana",
        "Times New Roman",
        "Comic Sans MS",
        "Fixedsys",
        "MS Sans Serif",
        "MS Serif",
        "Symbol",
        "System")

    for font in fonts:
        Radiobutton(self.top, text=font, variable=self.fontvar,
                                    value=font).pack(anchor=W) #creates the radio buttons
    frame = Frame(self.top)      #creates a frame in top
    frame.pack()                 #packs the frame
    applyButton = Button(frame, text="Apply", command=self.applyfont) #creates a button
                                            which when clicked calls the function applyfont
    applyButton.pack(side=LEFT)
    acceptButton = Button(frame, text="Accept", command=self.applyfont_exit) #creates a
                                    button which when clicked calls the function applyfont
    acceptButton.pack(side=RIGHT)
```

9.1.2 Function to apply change

This function applies the changes to the text.

```
def applyfont(self):

    self.font = self.fontvar.get()       #changes the font to fontvar which stores the
                                          value of the selected radiobutton

    self.text.config(font=(self.font, self.fontsize, self.fontstyle)) #resets text
                                                                        configurations
```

9.1.3 Function to accept change

This function applies the changes and terminates the font dialog box.

```
def applyfont_exit(self):
    self.font = self.fontvar.get()
    self.text.config(font=(self.font, self.fontsize, self.fontstyle))
```

```
    self.top.destroy()                    #terminates top
```

9.2 Changing font style

The functions are more or less the same as the last set of functions. The changes are mentioned below.

```
def Fontstyle(self):
    self.top = Toplevel()
    self.top.title("Font Style")
    label = Label(self.top, text="Please select a font style...", width=30)
    label.pack()
    styles = (
            "Normal",              #creates a tuple containing different font styles
            "bold",
            "italic")
    for style in styles:
        Radiobutton(self.top, text=style, value=style,
variable=self.style).pack(anchor=W)

    frame = Frame(self.top)
    frame.pack()
    applyButton = Button(frame, text="Apply", command=self.applyfontstyle)
    applyButton.pack(side=LEFT)
    acceptButton = Button(frame, text="Accept", command=self.applyfontstyle_exit)
    acceptButton.pack(side=RIGHT)

def applyfontstyle(self):
    self.fontstyle = self.style.get()
    self.text.config(font=(self.font, self.fontsize, self.fontstyle))

def applyfontstyle_exit(self):
    self.fontstyle = self.style.get()
    self.text.config(font=(self.font, self.fontsize, self.fontstyle))
    self.top.destroy()
```

9.3 Changing font size

The functions are more or less the same as the last set of functions. The changes are mentioned below.

```
def Fontsize(self):
    self.top = Toplevel()
    self.top.title("Font Size")

    label = Label(self.top, text="Please select a font size...", width=30)
    label.pack()

    self.scale = Scale(self.top, from_=8, to=72, orient=HORIZONTAL) #creates a scale by
                                                    which the user can set the font size
    self.scale.pack()
    self.scale.set(self.fontsize)

    frame = Frame(self.top)
    frame.pack()
    applyButton = Button(frame, text="Apply", command=self.applyfontsize)
    applyButton.pack(side=LEFT)
    acceptButton = Button(frame, text="Accept", command=self.applyfontsize_exit)
    acceptButton.pack(side=RIGHT)

def applyfontsize(self):
    self.fontsize = self.scale.get()
    self.text.config(font=(self.font, self.fontsize, self.fontstyle))

def applyfontsize_exit(self):
    self.fontsize = self.scale.get()
    self.text.config(font=(self.font, self.fontsize, self.fontstyle))
    self.top.destroy()
```

10. About menu

Under this menu you will create a info..

```
def info(self):
    showinfo("Information","This a text editor made using python :)")
    #opens a messagebox showing the above information.
```

11. Creating the object

noteobj=notepad()

This is the last step of typing the code, which is to create the object of the class notepad. **Do not forget this step**.

12. Creating an executable file

After executing your code properly you may want to create an executable file which you will be able to use any time. To do the same follow these steps:

I. First of all you need to set the path for python->scripts folder, which is mentioned in section 4, to use the command pyinstaller.

II. Go to the folder where your python file (the notepad you just created) is located.
III. Right click anywhere in the folder while holding the shift key.
IV. A menu will appear. Select the open cmd window here option.
V. Cmd will appear. Write down this command: **pyinstaller -w -F yourfilename.py** and press the return key. After successful task close cmd.
VI. You will find the executable file in the **dist** folder.

Note: **-w** prevents a command window to appear below your exe file.
-F(capital F) helps to create only a single exe file and abstracts the other files.

13. Resources

You can access the code and the exe file here
https://github.com/thenath1998/pynote.git

If you want to learn more about python click here
http://learnstack.in/course/overview/programming-101-in-python

Created by
-Sumon Nath