Final Project Report

Algorithm and Programming (COMP6047001)

Notepad

Anisa Dzikra Qalbiah

2502043116

L1CC

Project Overview

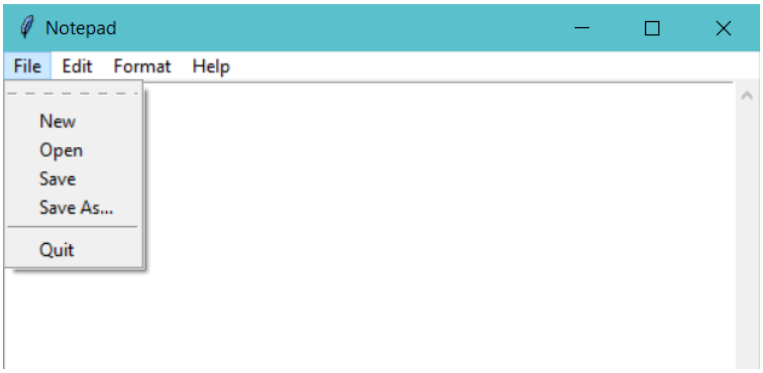
This project is a text editor program, where the user can take notes. The inspiration for this project came from how hard it is to take a fast note for a class, and from how other simple text editor program lack some sort of personalization for the user. This notepad can change font color and background color, which other simple text editor don't have. For me, this is a quite important feature, since sometimes you need to change color for some key notes

This notepad has some basic feature like usual "File Tab", "Edit Tab", "Format Tab", and "Help Tab". Some basic text formatting options such as "Font Type", "Font Size", "Bold", "Italic", "Underline", and "Strikethrough (overstrike)". I added an option to add date too, so it will be easy to mark the date of the note. I also add some color to the note (font color, background color) so it won't be a simple, boring, dull looking white note with black text.

Design

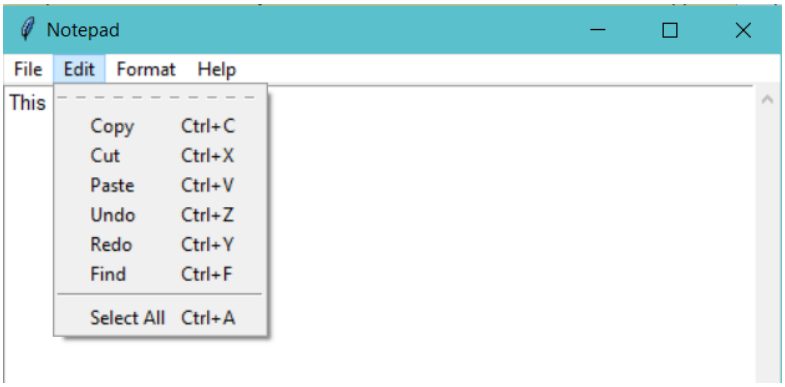
File Tab

This tab includes “New”, “Open”, “Save”, “Save As”, and the “Quit” option.



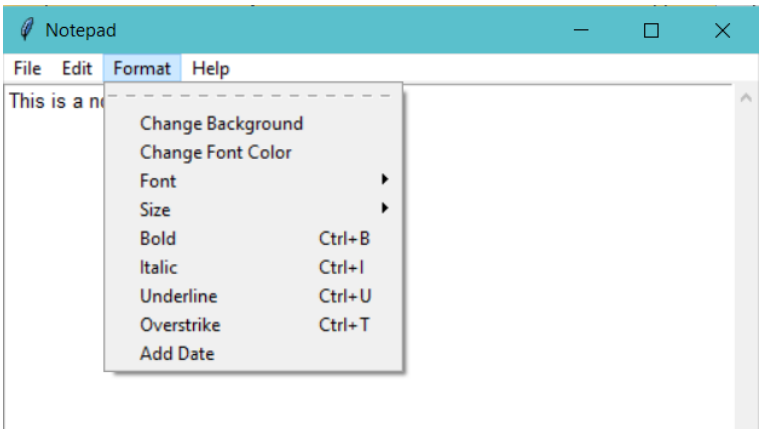
Edit Tab

This tab includes “Copy”, “Paste”, “Cut”, “Find”, “Undo”, “Redo”, and “Select All” with shortcut key.



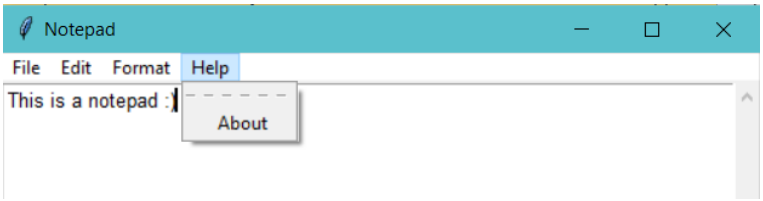
Format Tab

This tab includes “Change Background”, Change Font Color”, “Font”, “Size”, “Bold, “Italic”, “Underline”, “Overstrike”, and “Add Date”.



Help Tab

This tab includes “About”.



Implementation and Explanation of The Code

This project has 5 files, 1 main files, and 4 additional files for each tab options. This project uses module from Tkinter. There are 4 classes separated by files, each class for each tab options.

filetab.py

```
"""The file tab class"""
class File():
    def newFile(self):
        self.defaultfilename = "Untitled" #default file name
        self.text.delete(0.0, END)

    def saveFile(self):
        try:
            txt = self.text.get(0.0, END)
            f = open(self.filename, 'w') #saving file
            f.write(txt)
            f.close()
        except:
            self.saveAs()
```

Starting from the 'File' class, this class will include what kind of options you will get when you click the tab class.

newFile is used for when the user wanted a new clean window to work with. The .filename have "untitled" as the default name, and the .delete will clean the window.

saveFile is used to save the file the user is working with. The .write and .close signify that the user is saving the file.

```
def saveAs(self):
    f = asksaveasfile(mode='w', defaultextension='.txt') #save as option with default txt
    txt = self.text.get(0.0, END)
    try:
        f.write(txt.rstrip()) #rstrip() removes characters from the right based on the argument
        #(a string specifying the set of characters to be removed)
    except:
        showerror(title="Oops!", message="Unable to save file...")

def openFile(self):
    f = askopenfile(mode='r') #opening file
    self.defaultfilename = f.name
    txt = f.read()
    self.text.delete(0.0, END)
    self.text.insert(0.0, txt)
```

saveAs is used to save with the default extension of .txt. When it fails to save (the user closes the save as window) it will show an error that said "Unable to save file".

openFile is used when the user wanted to open a file that has been saved.

```
def quit(self):
    entry = askyesno(title="Quit", message="Are you sure you want to quit?")
    if entry == True:
        self.root.destroy() #destroy all widgets and exit mainloop

def __init__(self, text, root):
    self.defaultfilename = None
    self.text = text
    self.root = root
```

quit is used when the user wanted to quit using the quit option, the program will ask if the user wanted to quit or no, if the user say yes then the program will destroy all the widgets and exit the mainloop.

__init__ method to initialize the whole .filename, .text, and .root.

```
def main(root, text, menubar):
    filemenu = Menu(menubar)
    comFile = File(text, root)
    #adding command, label
    filemenu.add_command(label="New", command=comFile.newFile)
    filemenu.add_command(label="Open", command=comFile.openFile)
    filemenu.add_command(label="Save", command=comFile.saveFile)
    filemenu.add_command(label="Save As...", command=comFile.saveAs)
    filemenu.add_separator()
    filemenu.add_command(label="Quit", command=comFile.quit)
    menubar.add_cascade(label="File", menu=filemenu)
    root.config(menu=menubar)
```

The main method is used for calling the menubar for the filetab.

The add_command method is used to command the program to do what the user want.
The add_separator method is used to add a separator line for the whole tab.

The add_cascade creates a new hierarchical menu by associating a given menu to parent menu.

Config() is used to access an object's attributes after the initialization.

edittab.py

```
"""The edit tab class"""
class Edit():
    def popup(self, event):
        self.rightClick.post(event.x_root, event.y_root) #procedure posts a menu at a given position on the screen

    def copy(self, *args): #*args allows you to do is take in more arguments than the number of formal arguments that you previously defined
        select = self.word.selection_get() #get text that has been selected
        self.clipboard = select

    def paste(self, *args):
        self.word.insert(INSERT, self.clipboard) #insert text that has been copied
```

The ‘edit’ class will include what kind of options you will get when you click the tab class.

The rightClick.post in popup method is used to “post” a menu at a given position.

The copy method uses *args to allow to take in more arguments, with the selection_get() to select the text that has been selected. Clipboard is used to store data when copying.

The paste method uses insert to insert some string which in this case is a text from the clipboard that has been copied.

```
def cut(self, *args):
    select = self.word.selection_get()
    self.clipboard = select
    self.word.delete(SEL_FIRST, SEL_LAST) #delete text

def selectAll(self, *args):
    self.word.tag_add(SEL, "1.0", END) #the method tags either the position defined by startindex, or a range delimited by the positions
    self.word.mark_set(0.0, END) #informs a new position to the given mark
    self.word.see(INSERT) #this method returns true if the text located at the index position is visible

def undo(self, *args):
    self.word.edit_undo() #undoes the last edit action

def redo(self, *args):
    self.word.edit_redo() #redoes the last edit action
```

Cut is used when the user wanted to delete the text or words that has been selected, by using the delete() function, the text will get deleted.

selectAll method is when the user wanted to select all of the text on the noted, from the very start to the “END”.

undo method is to undoes the last edit action that the user did. Using edit_undo() the program will undo the last action.

Redo method is to redoes the last edit action that the user did. Using edit_redo() the program will redo the last action.

```
def find(self, *args):
    self.word.tag_remove('found', '1.0', END) #removes the tag
    target = askstring('Find', 'Search word:') #provide dialogs that prompt the user to enter a value of the desired type

    if target:
        idx = '1.0'
        while 1:
            idx = self.word.search(target, idx, nocase=1, stopindex=END) #searches pattern, stop index to limit the search

            if not idx: break
            lastidx = '%s+%dc' % (idx, len(target))

            self.word.tag_add('found', idx, lastidx)
            idx = lastidx
        self.word.tag_config('found', foreground='white', background='blue') #to change the value of options for the tag when
        #foreground is the color used for text, background is the color used for the text with this tag

def __init__(self, word, root):
    self.clipboard = None
    self.word = word
    self.rightClick = Menu(root)
```

The find method is used when the user wanted to search for some specific word or text. First the program ask the user to input the word or text, then the program will search for some “pattern” then stop if there is no text, but if the text is there, the program will highlight it to blue.

__init__ method to initialize the whole .clipboard, .word, and .rightClick.

```
def main(root, word, menubar):

    comEdit = Edit(word, root)

    editmenu = Menu(menubar)
    #adding command, label, and the shortcut
    editmenu.add_command(label="Copy", command=comEdit.copy, accelerator="Ctrl+C")
    editmenu.add_command(label="Paste", command=comEdit.paste, accelerator="Ctrl+V")
    editmenu.add_command(label="Cut", command=comEdit.cut, accelerator="Ctrl+X")
    editmenu.add_command(label="Undo", command=comEdit.undo, accelerator="Ctrl+Z")
    editmenu.add_command(label="Redo", command=comEdit.redo, accelerator="Ctrl+Y")
    editmenu.add_command(label="Find", command=comEdit.find, accelerator="Ctrl+F")
    editmenu.add_separator()
    editmenu.add_command(label="Select All", command=comEdit.selectAll, accelerator="Ctrl+A")
    menubar.add_cascade(label="Edit", menu=editmenu)

    root.bind_all("<Control-z>", comEdit.undo)
    root.bind_all("<Control-y>", comEdit.redo)
    root.bind_all("<Control-f>", comEdit.find)
    root.bind_all("<Control-a>", comEdit.selectAll)
```

The main method is used for calling the menubar for the edit tab.

The add_command method is used to command the program to do what the user want.
The add_separator method is used to add a separator line for the whole tab.

The add_cascade creates a new hierarchical menu by associating a given menu to parent menu.

Bind_all is used to bind some keys and call some functions.

```
comEdit.rightClick.add_command(label="Copy", command=comEdit.copy)
comEdit.rightClick.add_command(label="Cut", command=comEdit.cut)
comEdit.rightClick.add_command(label="Paste", command=comEdit.paste)
comEdit.rightClick.add_separator()
comEdit.rightClick.add_command(label="Select All", command=comEdit.selectAll)
comEdit.rightClick.bind("<Control-q>", comEdit.selectAll)

word.bind("<Button-3>", comEdit.popup)

root.config(menu=menubar)
```

This rightClick command is when the user right click their mouse and the option will show up.

The .bind will put the popup when the user right click.

Config() is used to access an object's attributes after the initialization.

formattab.py

```
"""The format tab class"""
class Format():
    def __init__(self, word):
        self.word = word

    def changeBackgroundColor(self):
        (triple, hexstr) = askcolor() #converts a number or binary data value to text formatted as a hexadecimal number
        if hexstr:  #converts a number to text formatted as a hexadecimal number
            self.word.config(bg=hexstr)

    def changeFontColor(self):
        (triple, hexstr) = askcolor() #converts a number or binary data value to text formatted as a hexadecimal number
        if hexstr: #converts a number to text formatted as a hexadecimal number
            self.word.config(fg=hexstr)
```

The 'format' class will include what kind of options you will get when you click the tab class.

__init__ method to initialize word. changeBackgroundColor and changeFontColor is a method where the user will choose a color from the color palette and then the program will configure.

```
def bold(self, *args): #bold text option
    try:
        current_tags = self.word.tag_names("sel.first")
        if "bold" in current_tags:
            self.word.tag_remove("bold", "sel.first", "sel.last")
        else:
            self.word.tag_add("bold", "sel.first", "sel.last")
            bold_font = Font(self.word, self.word.cget("font")) #cget method to get text option value of label
            bold_font.configure(weight="bold") #query or modify the options for a specified tagname
            self.wordt.tag_configure("bold", font=bold_font)
    except:
        pass
```

Bold is an option for the text format, here the tag_names method will return a sequence of all the tag names that are associated, then the tag_remove will remove the tagname from all characters between “sel.first” and “sel.last”.

The tag_add associates the tag named with a region starting just after “sel.first” and extending into “sel.last”. If you omit “sel.last”, only the character after “sel.first” is tagged

The tag_cget is to retrieve the value of the given option for the tag.

The tag_configure is to change the value of options for the tag, pass in one or more (options) pairs.

And it is the same for the *italic*, underline, and the ~~strikethrough~~.

```
def italic(self, *args): #italic text option
    try:
        current_tags = self.word.tag_names("sel.first")
        if "italic" in current_tags:
            self.word.tag_remove("italic", "sel.first", "sel.last")
        else:
            self.word.tag_add("italic", "sel.first", "sel.last")
            italic_font = Font(self.word, self.word.cget("font"))
            italic_font.configure(slant="italic")
            self.word.tag_configure("italic", font=italic_font)
    except:
        pass
```

```
def underline(self, *args): #underline word option
    try:
        current_tags = self.word.tag_names("sel.first")
        if "underline" in current_tags:
            self.word.tag_remove("underline", "sel.first", "sel.last")
        else:
            self.word.tag_add("underline", "sel.first", "sel.last")
            underline_font = Font(self.word, self.word.cget("font"))
            underline_font.configure(underline=1)
            self.word.tag_configure("underline", font=underline_font)
    except:
        pass
```

```
def overstrike(self, *args):   #overstrike word option
    try:
        current_tags = self.word.tag_names("sel.first")
        if "overstrike" in current_tags:
            self.word.tag_remove("overstrike", "sel.first", "sel.last")
        else:
            self.word.tag_add("overstrike", "sel.first", "sel.last")
            overstrike_font = Font(self.word, self.word.cget("font"))
            overstrike_font.configure(overstrike=1)
            self.word.tag_configure("overstrike", font=overstrike_font)
    except:
        pass
```

```
def addDate(self): #adding date to note
    full_date = time.localtime()
    day = str(full_date.tm_mday) #day
    month = str(full_date.tm_mon) #month
    year = str(full_date.tm_year) #year
    date = day + '/' + month + '/' + year #adding all together
    self.word.insert(INSERT, date, "a")
```

This addDate method is used to add dates when the note was taken to the note. From import time, we can get the localtime(), day, month, and year. Then on the “date”, we put all the day, month, and year together with the “/” as separator.

```
def main(root, word, menubar):
    comFormat = Format(word)

    fontoptions = families(root) #default font and size
    font = Font(family="Arial", size=10)
    word.configure(font=font)

    formatMenu = Menu(menubar)

    fontsubmenu = Menu(formatMenu, tearoff=0)
    sizesubmenu = Menu(formatMenu, tearoff=0)

    for option in fontoptions: #font options
        fontsubmenu.add_command(label=option, command=lambda option=option: font.configure(family=option))
    for value in range(1, 31): #size options
        sizesubmenu.add_command(label=str(value), command=lambda value=value: font.configure(size=value))
```

The main method is used for calling the menubar for the format tab.

Here the default font option is set to “arial” and with the size of 10.

The tearoff function is to detach the menu to be a floating menu.

For the fontoptions the floating menu consist of many font options, while for the value font size it is available from size 1 until 31, the label will take str(value) when the user clicks it.

```
#adding command, label, and the shortcut
formatMenu.add_command(label="Change Background", command=comFormat.changeBackgroundColor)
formatMenu.add_command(label="Change Font Color", command=comFormat.changeFontColor)
formatMenu.add_cascade(label="Font", underline=0, menu=fontsubmenu)
formatMenu.add_cascade(label="Size", underline=0, menu=sizesubmenu)
formatMenu.add_command(label="Bold", command=comFormat.bold, accelerator="Ctrl+B")
formatMenu.add_command(label="Italic", command=comFormat.italic, accelerator="Ctrl+I")
formatMenu.add_command(label="Underline", command=comFormat.underline, accelerator="Ctrl+U")
formatMenu.add_command(label="Overstrike", command=comFormat.overstrike, accelerator="Ctrl+T")
formatMenu.add_command(label="Add Date", command=comFormat.addDate)
menubar.add_cascade(label="Format", menu=formatMenu)

root.bind_all("<Control-b>", comFormat.bold)
root.bind_all("<Control-i>", comFormat.italic)
root.bind_all("<Control-u>", comFormat.underline)
root.bind_all("<Control-T>", comFormat.overstrike)

root.grid_columnconfigure(0, weight=1)
root.resizable(True, True)

root.config(menu=menubar)
```

The add_command method is used to command the program to do what the user want.

The add_cascade creates a new hierarchical menu by associating a given menu to parent menu.

Bind_all is used to bind some keys and call some functions.

The grid_columnfigure is to configure column index of a grid, while the resizable() is for the window to change its size according to the user.

helptab.py

```
"""The help tab class"""
class Help():
    def about(root):
        showinfo(title="About", message=f"This a notepad :))")


def main(root, text, menubar):

    help = Help()

    helpMenu = Menu(menubar)
    helpMenu.add_command(label="About", command=help.about)
    menubar.add_cascade(label="Help", menu=helpMenu)

    root.config(menu=menubar)
```

Last but not least is the ‘help’ class. The ‘help’ class will include what kind of options you will get when you click the tab class.

Here when the user clicks the help tab, the “about” option will show up and when the user clicks it, the message “This is a notepad (smiley face)” will show up.

The main method is used for calling the menubar for the help tab.

The add_command method is used to command the program to do what the user want.

The add_cascade creates a new hierarchical menu by associating a given menu to parent menu.

References and Helpful Links

https://www.tutorialspoint.com/python/tk_menu.htm

https://docs.python.org/3/library/dialog.html

https://www.programcreek.com/python/example/95889/tkinter.filedialog.asksaveasfilehttps://www.geeksforgeeks.org/right-click-menu-using-tkinter/

https://stackoverflow.com/questions/67433258/tkinter-undo-and-redo-the-formattings-made-to-a-text-widget

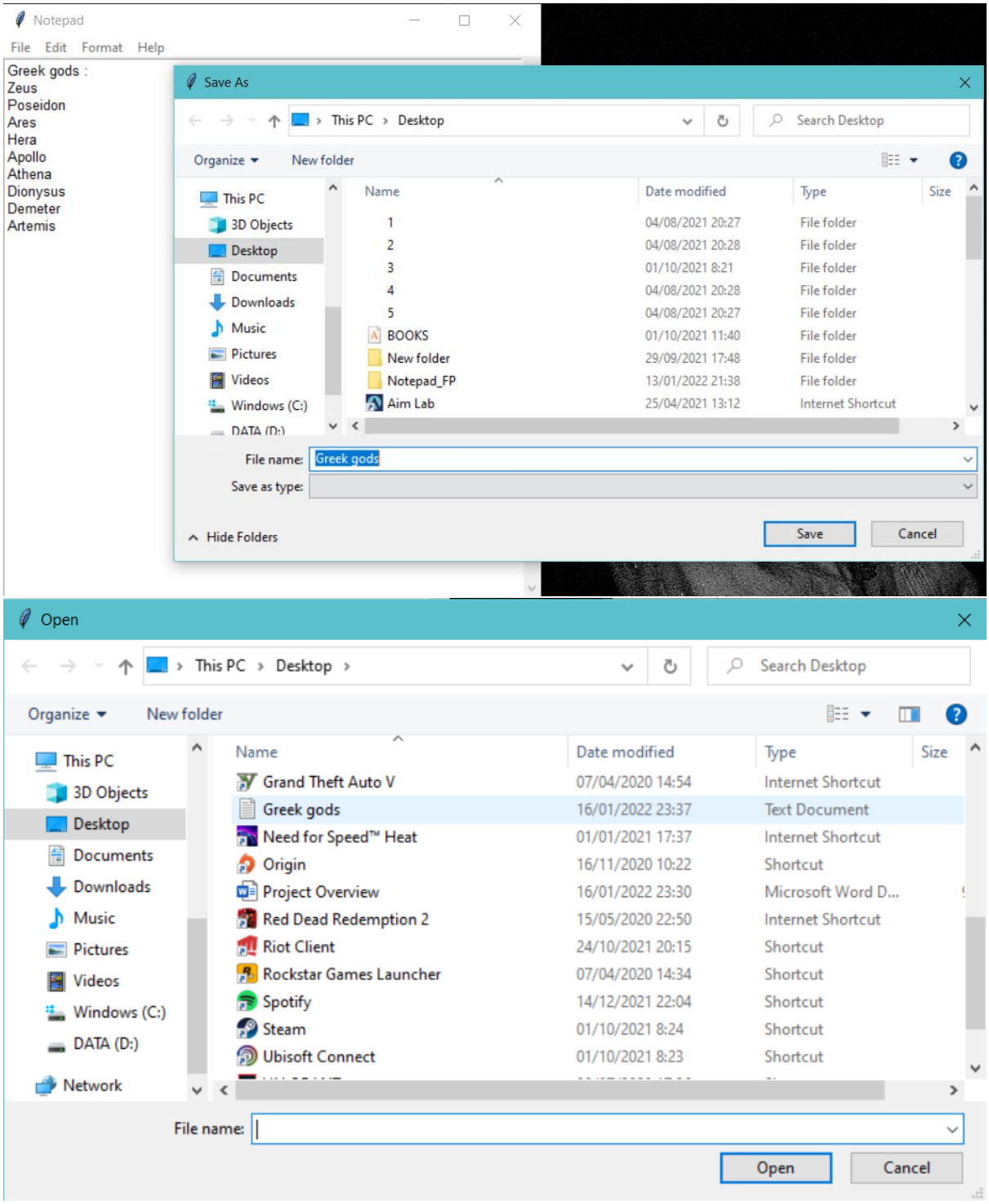
https://www.tutorialspoint.com/python/tk_text.htm

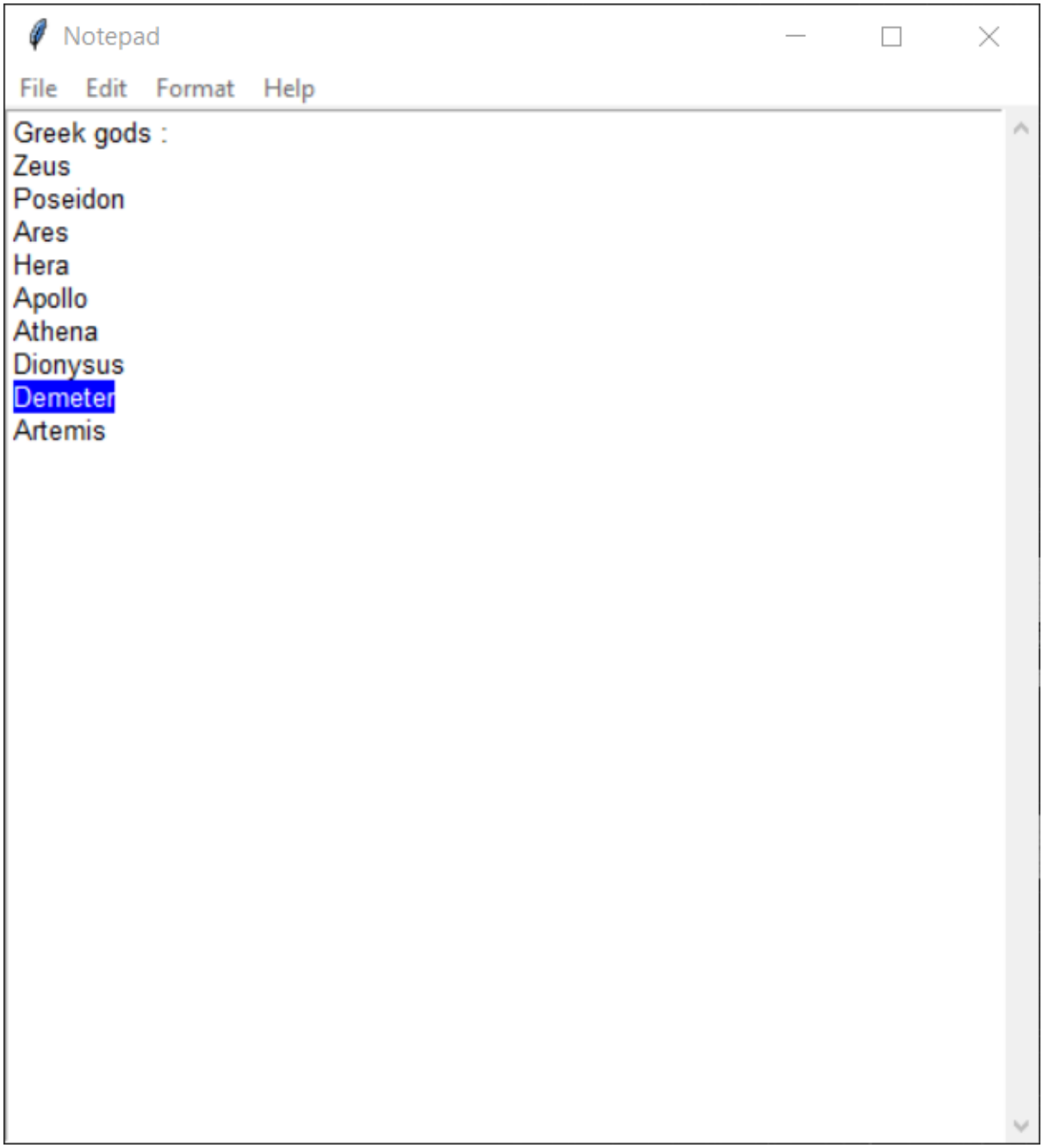
https://docstore.mik.ua/orelly/other/python/0596001886_pythonian-chp-16-sect-6.html

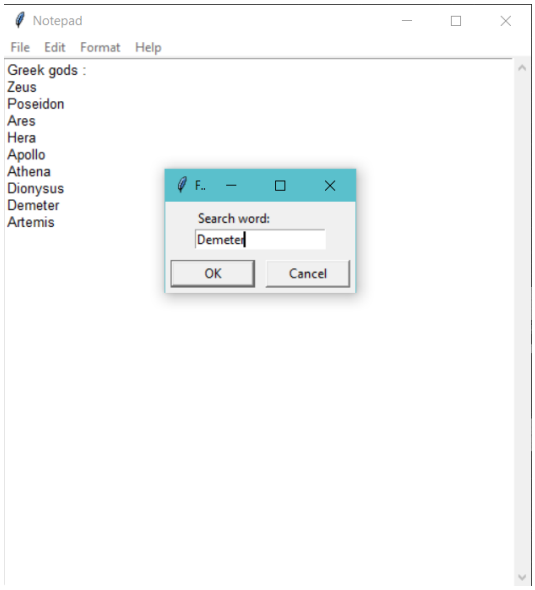
https://pythonguides.com/python-tkinter-search-box/

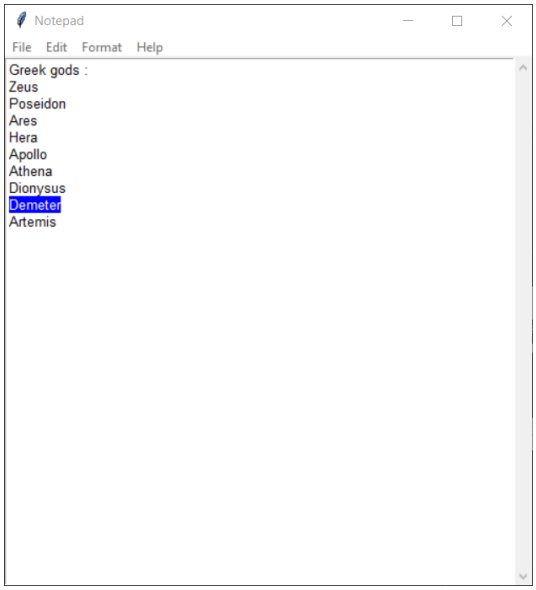
https://www.tutorialexample.com/python-tkinter-bind-ctrlkey-python-tutorial/

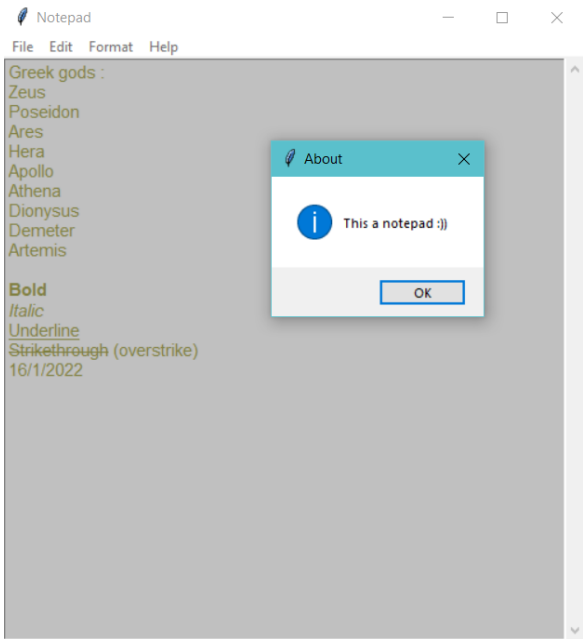
Pictures of The Program

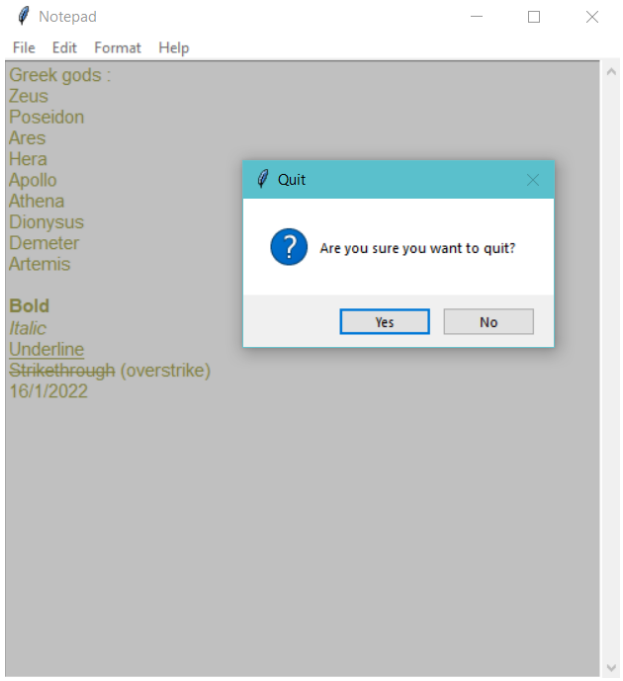



Notepad
File Edit Format Help
Greek gods :
Zeus
Poseidon
Ares
Hera
Apollo
Athena
Dionysus
Demeter
Artemis


Notepad
File Edit Format Help
Greek gods :
Zeus
Poseidon
Ares
Hera
Apollo
Athena
Dionysus
Demeter
Artemis
F..
Search word:
Demeter
OK
Cancel


Notepad
File Edit Format Help
Greek gods :
Zeus
Poseidon
Ares
Hera
Apollo
Athena
Dionysus
Demeter
Artemis


Notepad
File Edit Format Help
Greek gods :
Zeus
Poseidon
Ares
Hera
Apollo
Athena
Dionysus
Demeter
Artemis
Bold
Italic
Underline
Strikethrough (overstrike)
16/1/2022
About
This a notepad :))
OK


Notepad
File Edit Format Help
Greek gods :
Zeus
Poseidon
Ares
Hera
Apollo
Athena
Dionysus
Demeter
Artemis
Bold
Italic
Underline
Strikethrough (overstrike)
16/1/2022
Quit
Are you sure you want to quit?
Yes
No