import sys

import speech\_recognition as sr

import pyaudio

from PyQt5 import QtWidgets

from PyQt5 import QtPrintSupport

from PyQt5 import QtGui, QtCore

from PyQt5.QtCore import Qt

from ext import \*

class Main(QtWidgets.QMainWindow):

def \_\_init\_\_(self,parent=None):

QtWidgets.QMainWindow.\_\_init\_\_(self,parent)

self.filename = ""

self.changesSaved = True

self.initUI()

def initToolbar(self):

self.newAction = QtWidgets.QAction(QtGui.QIcon("new.ico"),"New",self)

self.newAction.setShortcut("Ctrl+N")

self.newAction.setStatusTip("Create a new document from scratch.")

self.newAction.triggered.connect(self.new)

self.openAction = QtWidgets.QAction(QtGui.QIcon("open.ico"),"Open file",self)

self.openAction.setStatusTip("Open existing document")

self.openAction.setShortcut("Ctrl+O")

self.openAction.triggered.connect(self.open)

self.saveAction = QtWidgets.QAction(QtGui.QIcon("save.ico"),"Save",self)

self.saveAction.setStatusTip("Save document")

self.saveAction.setShortcut("Ctrl+S")

self.saveAction.triggered.connect(self.save)

self.printAction = QtWidgets.QAction(QtGui.QIcon("print.ico"),"Print document",self)

self.printAction.setStatusTip("Print document")

self.printAction.setShortcut("Ctrl+P")

self.printAction.triggered.connect(self.printHandler)

self.previewAction = QtWidgets.QAction(QtGui.QIcon("page\_view.ico"),"Page view",self)

self.previewAction.setStatusTip("Preview page before printing")

self.previewAction.setShortcut("Ctrl+Shift+P")

self.previewAction.triggered.connect(self.preview)

self.findAction = QtWidgets.QAction(QtGui.QIcon("find\_replace.ico"),"Find and replace",self)

self.findAction.setStatusTip("Find and replace words in your document")

self.findAction.setShortcut("Ctrl+F")

self.findAction.triggered.connect(find.Find(self).show)

self.cutAction = QtWidgets.QAction(QtGui.QIcon("cut.ico"),"Cut to clipboard",self)

self.cutAction.setStatusTip("Delete and copy text to clipboard")

self.cutAction.setShortcut("Ctrl+X")

self.cutAction.triggered.connect(self.text.cut)

self.copyAction = QtWidgets.QAction(QtGui.QIcon("copy.ico"),"Copy to clipboard",self)

self.copyAction.setStatusTip("Copy text to clipboard")

self.copyAction.setShortcut("Ctrl+C")

self.copyAction.triggered.connect(self.text.copy)

self.pasteAction = QtWidgets.QAction(QtGui.QIcon("paste.ico"),"Paste from clipboard",self)

self.pasteAction.setStatusTip("Paste text from clipboard")

self.pasteAction.setShortcut("Ctrl+V")

self.pasteAction.triggered.connect(self.text.paste)

self.undoAction = QtWidgets.QAction(QtGui.QIcon("undo.ico"),"Undo last action",self)

self.undoAction.setStatusTip("Undo last action")

self.undoAction.setShortcut("Ctrl+Z")

self.undoAction.triggered.connect(self.text.undo)

self.redoAction = QtWidgets.QAction(QtGui.QIcon("redo.ico"),"Redo last undone thing",self)

self.redoAction.setStatusTip("Redo last undone thing")

self.redoAction.setShortcut("Ctrl+Y")

self.redoAction.triggered.connect(self.text.redo)

dateTimeAction = QtWidgets.QAction(QtGui.QIcon("date\_time.ico"),"Insert current date/time",self)

dateTimeAction.setStatusTip("Insert current date/time")

dateTimeAction.setShortcut("Ctrl+D")

dateTimeAction.triggered.connect(datetime.DateTime(self).show)

wordCountAction = QtWidgets.QAction(QtGui.QIcon("word\_symbol\_count.ico"),"See word/symbol count",self)

wordCountAction.setStatusTip("See word/symbol count")

wordCountAction.setShortcut("Ctrl+W")

wordCountAction.triggered.connect(self.wordCount)

tableAction = QtWidgets.QAction(QtGui.QIcon("table.ico"),"Insert table",self)

tableAction.setStatusTip("Insert table")

tableAction.setShortcut("Ctrl+T")

tableAction.triggered.connect(table.Table(self).show)

imageAction = QtWidgets.QAction(QtGui.QIcon("image.ico"),"Insert image",self)

imageAction.setStatusTip("Insert image")

imageAction.setShortcut("Ctrl+I")

imageAction.triggered.connect(self.insertImage)

bulletAction = QtWidgets.QAction(QtGui.QIcon("bulleted.ico"),"Insert bullet List",self)

bulletAction.setStatusTip("Insert bullet list")

bulletAction.setShortcut("Ctrl+Shift+B")

bulletAction.triggered.connect(self.bulletList)

numberedAction = QtWidgets.QAction(QtGui.QIcon("numbered.ico"),"Insert numbered List",self)

numberedAction.setStatusTip("Insert numbered list")

numberedAction.setShortcut("Ctrl+Shift+L")

numberedAction.triggered.connect(self.numberList)

speechRecog = QtWidgets.QAction(QtGui.QIcon('mic.ico'), 'Speech Notes', self)

speechRecog.setStatusTip('Speak something in your mic')

speechRecog.setShortcut('Ctrl+M')

speechRecog.triggered.connect(self.speechRecog)

self.toolbar = self.addToolBar("Options")

self.toolbar.addAction(self.newAction)

self.toolbar.addAction(self.openAction)

self.toolbar.addAction(self.saveAction)

self.toolbar.addSeparator()

self.toolbar.addAction(self.printAction)

self.toolbar.addAction(self.previewAction)

self.toolbar.addSeparator()

self.toolbar.addAction(self.cutAction)

self.toolbar.addAction(self.copyAction)

self.toolbar.addAction(self.pasteAction)

self.toolbar.addAction(self.undoAction)

self.toolbar.addAction(self.redoAction)

self.toolbar.addSeparator()

self.toolbar.addAction(self.findAction)

self.toolbar.addAction(dateTimeAction)

self.toolbar.addAction(wordCountAction)

self.toolbar.addAction(tableAction)

self.toolbar.addAction(imageAction)

self.toolbar.addSeparator()

self.toolbar.addAction(bulletAction)

self.toolbar.addAction(numberedAction)

self.toolbar.addSeparator()

self.toolbar.addAction(speechRecog)

self.addToolBarBreak()

def initFormatbar(self):

fontBox = QtWidgets.QFontComboBox(self)

fontBox.currentFontChanged.connect(lambda font: self.text.setCurrentFont(font))

fontSize = QtWidgets.QSpinBox(self)

# Will display " pt" after each value

fontSize.setSuffix(" pt")

fontSize.valueChanged.connect(lambda size: self.text.setFontPointSize(size))

fontSize.setValue(14)

fontColor = QtWidgets.QAction(QtGui.QIcon("text\_color.ico"),"Change font color",self)

fontColor.triggered.connect(self.fontColorChanged)

boldAction = QtWidgets.QAction(QtGui.QIcon("bold.ico"),"Bold",self)

boldAction.setStatusTip('Make a text appear bold')

boldAction.setShortcut("Ctrl+B")

boldAction.triggered.connect(self.bold)

italicAction = QtWidgets.QAction(QtGui.QIcon("italic.ico"),"Italic",self)

italicAction.setStatusTip('Make a text appear Italic')

italicAction.setShortcut("Ctrl+I")

italicAction.triggered.connect(self.italic)

underlAction = QtWidgets.QAction(QtGui.QIcon("underline.ico"),"Underline",self)

underlAction.setStatusTip('Make a text appear underlined')

underlAction.setShortcut("Ctrl+U")

underlAction.triggered.connect(self.underline)

strikeAction = QtWidgets.QAction(QtGui.QIcon("strike\_through.ico"),"Strike-out",self)

strikeAction.setStatusTip('Make a text appear Striked out')

strikeAction.triggered.connect(self.strike)

superAction = QtWidgets.QAction(QtGui.QIcon("superscript.ico"),"Superscript",self)

superAction.setStatusTip('Make a text superscripted')

superAction.setShortcut("Ctrl+PgUp")

superAction.triggered.connect(self.superScript)

subAction = QtWidgets.QAction(QtGui.QIcon("subscript.ico"),"Subscript",self)

subAction.setStatusTip('Make a text subscripted')

subAction.setShortcut("Ctrl+PgDn")

subAction.triggered.connect(self.subScript)

alignLeft = QtWidgets.QAction(QtGui.QIcon("align\_left.png"),"Align left",self)

alignLeft.setStatusTip('Align a text to the left')

alignLeft.setShortcut("Ctrl+L")

alignLeft.triggered.connect(self.alignLeft)

alignCenter = QtWidgets.QAction(QtGui.QIcon("align\_center.png"),"Align center",self)

alignCenter.setStatusTip('Align a text to the center')

alignCenter.setShortcut("Ctrl+E")

alignCenter.triggered.connect(self.alignCenter)

alignRight = QtWidgets.QAction(QtGui.QIcon("align\_right.png"),"Align right",self)

alignRight.setStatusTip('Align a text to the right')

alignRight.setShortcut("Ctrl+R")

alignRight.triggered.connect(self.alignRight)

alignJustify = QtWidgets.QAction(QtGui.QIcon("align\_justify.png"),"Align justify",self)

alignJustify.setStatusTip('Align a text justified')

alignJustify.setShortcut("Ctrl+J")

alignJustify.triggered.connect(self.alignJustify)

indentAction = QtWidgets.QAction(QtGui.QIcon("indent.ico"),"Indent Area",self)

indentAction.setStatusTip('Make a text indented')

indentAction.setShortcut("Ctrl+Tab")

indentAction.triggered.connect(self.indent)

dedentAction = QtWidgets.QAction(QtGui.QIcon("dedent.ico"),"Dedent Area",self)

dedentAction.setStatusTip('Make a text dedented')

dedentAction.setShortcut("Shift+Tab")

dedentAction.triggered.connect(self.dedent)

backColor = QtWidgets.QAction(QtGui.QIcon("background\_color.ico"),"Change background color",self)

backColor.triggered.connect(self.highlight)

self.formatbar = self.addToolBar("Format")

self.formatbar.addWidget(fontBox)

self.formatbar.addWidget(fontSize)

self.formatbar.addSeparator()

self.formatbar.addAction(fontColor)

self.formatbar.addAction(backColor)

self.formatbar.addSeparator()

self.formatbar.addAction(boldAction)

self.formatbar.addAction(italicAction)

self.formatbar.addAction(underlAction)

self.formatbar.addAction(strikeAction)

self.formatbar.addAction(superAction)

self.formatbar.addAction(subAction)

self.formatbar.addSeparator()

self.formatbar.addAction(alignLeft)

self.formatbar.addAction(alignCenter)

self.formatbar.addAction(alignRight)

self.formatbar.addAction(alignJustify)

self.formatbar.addSeparator()

self.formatbar.addAction(indentAction)

self.formatbar.addAction(dedentAction)

def initMenubar(self):

menubar = self.menuBar()

file = menubar.addMenu("File")

edit = menubar.addMenu("Edit")

view = menubar.addMenu("View")

# Add the most important actions to the menubar

file.addAction(self.newAction)

file.addAction(self.openAction)

file.addAction(self.saveAction)

file.addAction(self.printAction)

file.addAction(self.previewAction)

edit.addAction(self.undoAction)

edit.addAction(self.redoAction)

edit.addAction(self.cutAction)

edit.addAction(self.copyAction)

edit.addAction(self.pasteAction)

edit.addAction(self.findAction)

# Toggling actions for the various bars

toolbarAction = QtWidgets.QAction("Toggle Toolbar",self)

toolbarAction.triggered.connect(self.toggleToolbar)

formatbarAction = QtWidgets.QAction("Toggle Formatbar",self)

formatbarAction.triggered.connect(self.toggleFormatbar)

statusbarAction = QtWidgets.QAction("Toggle Statusbar",self)

statusbarAction.triggered.connect(self.toggleStatusbar)

view.addAction(toolbarAction)

view.addAction(formatbarAction)

view.addAction(statusbarAction)

def initUI(self):

self.text = QtWidgets.QTextEdit(self)

# Set the tab stop width to around 33 pixels which is

# more or less 8 spaces

self.text.setTabStopWidth(33)

self.initToolbar()

self.initFormatbar()

self.initMenubar()

self.setCentralWidget(self.text)

# Initialize a statusbar for the window

self.statusbar = self.statusBar()

# If the cursor position changes, call the function that displays

# the line and column number

self.text.cursorPositionChanged.connect(self.cursorPosition)

# We need our own context menu for tables

self.text.setContextMenuPolicy(Qt.CustomContextMenu)

self.text.customContextMenuRequested.connect(self.context)

self.text.textChanged.connect(self.changed)

self.setGeometry(100,100,1030,800)

self.setWindowTitle("Notepedi | Word Processor")

self.setWindowIcon(QtGui.QIcon("icons/icon.png"))

def changed(self):

self.changesSaved = False

def closeEvent(self,event):

if self.changesSaved:

event.accept()

else:

popup = QtWidgets.QMessageBox(self)

popup.setIcon(QtWidgets.QMessageBox.Warning)

popup.setText("The document has been modified")

popup.setInformativeText("Do you want to save your changes?")

popup.setStandardButtons(QtWidgets.QMessageBox.Save |

QtWidgets.QMessageBox.Cancel |

QtWidgets.QMessageBox.Discard)

popup.setDefaultButton(QtWidgets.QMessageBox.Save)

answer = popup.exec\_()

if answer == QtWidgets.QMessageBox.Save:

self.save()

elif answer == QtWidgets.QMessageBox.Discard:

event.accept()

else:

event.ignore()

def context(self,pos):

# Grab the cursor

cursor = self.text.textCursor()

# Grab the current table, if there is one

table = cursor.currentTable()

# Above will return 0 if there is no current table, in which case

# we call the normal context menu. If there is a table, we create

# our own context menu specific to table interaction

if table:

menu = QtGui.QMenu(self)

appendRowAction = QtWidgets.QAction("Append row",self)

appendRowAction.triggered.connect(lambda: table.appendRows(1))

appendColAction = QtWidgets.QAction("Append column",self)

appendColAction.triggered.connect(lambda: table.appendColumns(1))

removeRowAction = QtWidgets.QAction("Remove row",self)

removeRowAction.triggered.connect(self.removeRow)

removeColAction = QtWidgets.QAction("Remove column",self)

removeColAction.triggered.connect(self.removeCol)

insertRowAction = QtWidgets.QAction("Insert row",self)

insertRowAction.triggered.connect(self.insertRow)

insertColAction = QtWidgets.QAction("Insert column",self)

insertColAction.triggered.connect(self.insertCol)

mergeAction = QtWidgets.QAction("Merge cells",self)

mergeAction.triggered.connect(lambda: table.mergeCells(cursor))

# Only allow merging if there is a selection

if not cursor.hasSelection():

mergeAction.setEnabled(False)

splitAction = QtWidgets.QAction("Split cells",self)

cell = table.cellAt(cursor)

# Only allow splitting if the current cell is larger

# than a normal cell

if cell.rowSpan() > 1 or cell.columnSpan() > 1:

splitAction.triggered.connect(lambda: table.splitCell(cell.row(),cell.column(),1,1))

else:

splitAction.setEnabled(False)

menu.addAction(appendRowAction)

menu.addAction(appendColAction)

menu.addSeparator()

menu.addAction(removeRowAction)

menu.addAction(removeColAction)

menu.addSeparator()

menu.addAction(insertRowAction)

menu.addAction(insertColAction)

menu.addSeparator()

menu.addAction(mergeAction)

menu.addAction(splitAction)

# Convert the widget coordinates into global coordinates

pos = self.mapToGlobal(pos)

# Add pixels for the tool and formatbars, which are not included

# in mapToGlobal(), but only if the two are currently visible and

# not toggled by the user

if self.toolbar.isVisible():

pos.setY(pos.y() + 45)

if self.formatbar.isVisible():

pos.setY(pos.y() + 45)

# Move the menu to the new position

menu.move(pos)

menu.show()

else:

event = QtGui.QContextMenuEvent(QtGui.QContextMenuEvent.Mouse,QtCore.QPoint())

self.text.contextMenuEvent(event)

def removeRow(self):

# Grab the cursor

cursor = self.text.textCursor()

# Grab the current table (we assume there is one, since

# this is checked before calling)

table = cursor.currentTable()

# Get the current cell

cell = table.cellAt(cursor)

# Delete the cell's row

table.removeRows(cell.row(),1)

def removeCol(self):

# Grab the cursor

cursor = self.text.textCursor()

# Grab the current table (we assume there is one, since

# this is checked before calling)

table = cursor.currentTable()

# Get the current cell

cell = table.cellAt(cursor)

# Delete the cell's column

table.removeColumns(cell.column(),1)

def insertRow(self):

# Grab the cursor

cursor = self.text.textCursor()

# Grab the current table (we assume there is one, since

# this is checked before calling)

table = cursor.currentTable()

# Get the current cell

cell = table.cellAt(cursor)

# Insert a new row at the cell's position

table.insertRows(cell.row(),1)

def insertCol(self):

# Grab the cursor

cursor = self.text.textCursor()

# Grab the current table (we assume there is one, since

# this is checked before calling)

table = cursor.currentTable()

# Get the current cell

cell = table.cellAt(cursor)

# Insert a new row at the cell's position

table.insertColumns(cell.column(),1)

def toggleToolbar(self):

state = self.toolbar.isVisible()

# Set the visibility to its inverse

self.toolbar.setVisible(not state)

def toggleFormatbar(self):

state = self.formatbar.isVisible()

# Set the visibility to its inverse

self.formatbar.setVisible(not state)

def toggleStatusbar(self):

state = self.statusbar.isVisible()

# Set the visibility to its inverse

self.statusbar.setVisible(not state)

def new(self):

spawn = Main()

spawn.show()

def open(self):

# Get filename and show only .npd files

#PYQT5 Returns a tuple in PyQt5, we only need the filename

self.filename = QtWidgets.QFileDialog.getOpenFileName(self, 'Open File',".","(\*.npd)")[0]

if self.filename:

with open(self.filename,"rt") as file:

self.text.setText(file.read())

def save(self):

# Only open dialog if there is no filename yet

#PYQT5 Returns a tuple in PyQt5, we only need the filename

if not self.filename:

self.filename = QtWidgets.QFileDialog.getSaveFileName(self, 'Save File')[0]

if self.filename:

# Append extension if not there yet

if not self.filename.endswith(".npd"):

self.filename += ".npd"

# We just store the contents of the text file along with the

# format in html, which Qt does in a very nice way for us

with open(self.filename,"wt") as file:

file.write(self.text.toHtml())

self.changesSaved = True

def preview(self):

# Open preview dialog

preview = QtPrintSupport.QPrintPreviewDialog()

# If a print is requested, open print dialog

preview.paintRequested.connect(lambda p: self.text.print\_(p))

preview.exec\_()

def printHandler(self):

# Open printing dialog

dialog = QtPrintSupport.QPrintDialog()

if dialog.exec\_() == QtWidgets.QDialog.Accepted:

self.text.document().print\_(dialog.printer())

def cursorPosition(self):

cursor = self.text.textCursor()

# Mortals like 1-indexed things

line = cursor.blockNumber() + 1

col = cursor.columnNumber()

self.statusbar.showMessage("Line: {} | Column: {}".format(line,col))

def wordCount(self):

wc = wordcount.WordCount(self)

wc.getText()

wc.show()

def insertImage(self):

# Get image file name

#PYQT5 Returns a tuple in PyQt5

filename = QtWidgets.QFileDialog.getOpenFileName(self, 'Insert image',".","Images (\*.png \*.xpm \*.jpg \*.jpeg \*.bmp \*.gif)")[0]

if filename:

# Create image object

image = QtGui.QImage(filename)

# Error if unloadable

if image.isNull():

popup = QtWidgets.QMessageBox(QtWidgets.QMessageBox.Critical,

"Image load error",

"Could not load image file!",

QtWidgets.QMessageBox.Ok,

self)

popup.show()

else:

cursor = self.text.textCursor()

cursor.insertImage(image, filename)

cursor.image.scaledToHeight(100)

cursor.image.scaledToWidth(100)

def fontColorChanged(self):

# Get a color from the text dialog

color = QtWidgets.QColorDialog.getColor()

# Set it as the new text color

self.text.setTextColor(color)

def highlight(self):

color = QtWidgets.QColorDialog.getColor()

self.text.setTextBackgroundColor(color)

def bold(self):

if self.text.fontWeight() == QtGui.QFont.Bold:

self.text.setFontWeight(QtGui.QFont.Normal)

else:

self.text.setFontWeight(QtGui.QFont.Bold)

def italic(self):

state = self.text.fontItalic()

self.text.setFontItalic(not state)

def underline(self):

state = self.text.fontUnderline()

self.text.setFontUnderline(not state)

def strike(self):

# Grab the text's format

fmt = self.text.currentCharFormat()

# Set the fontStrikeOut property to its opposite

fmt.setFontStrikeOut(not fmt.fontStrikeOut())

# And set the next char format

self.text.setCurrentCharFormat(fmt)

def superScript(self):

# Grab the current format

fmt = self.text.currentCharFormat()

# And get the vertical alignment property

align = fmt.verticalAlignment()

# Toggle the state

if align == QtGui.QTextCharFormat.AlignNormal:

fmt.setVerticalAlignment(QtGui.QTextCharFormat.AlignSuperScript)

else:

fmt.setVerticalAlignment(QtGui.QTextCharFormat.AlignNormal)

# Set the new format

self.text.setCurrentCharFormat(fmt)

def subScript(self):

# Grab the current format

fmt = self.text.currentCharFormat()

# And get the vertical alignment property

align = fmt.verticalAlignment()

# Toggle the state

if align == QtGui.QTextCharFormat.AlignNormal:

fmt.setVerticalAlignment(QtGui.QTextCharFormat.AlignSubScript)

else:

fmt.setVerticalAlignment(QtGui.QTextCharFormat.AlignNormal)

# Set the new format

self.text.setCurrentCharFormat(fmt)

def alignLeft(self):

self.text.setAlignment(Qt.AlignLeft)

def alignRight(self):

self.text.setAlignment(Qt.AlignRight)

def alignCenter(self):

self.text.setAlignment(Qt.AlignCenter)

def alignJustify(self):

self.text.setAlignment(Qt.AlignJustify)

def indent(self):

# Grab the cursor

cursor = self.text.textCursor()

if cursor.hasSelection():

# Store the current line/block number

temp = cursor.blockNumber()

# Move to the selection's end

cursor.setPosition(cursor.anchor())

# Calculate range of selection

diff = cursor.blockNumber() - temp

direction = QtGui.QTextCursor.Up if diff > 0 else QtGui.QTextCursor.Down

# Iterate over lines (diff absolute value)

for n in range(abs(diff) + 1):

# Move to start of each line

cursor.movePosition(QtGui.QTextCursor.StartOfLine)

# Insert tabbing

cursor.insertText("\t")

# And move back up

cursor.movePosition(direction)

# If there is no selection, just insert a tab

else:

cursor.insertText("\t")

def handleDedent(self,cursor):

cursor.movePosition(QtGui.QTextCursor.StartOfLine)

# Grab the current line

line = cursor.block().text()

# If the line starts with a tab character, delete it

if line.startswith("\t"):

# Delete next character

cursor.deleteChar()

# Otherwise, delete all spaces until a non-space character is met

else:

for char in line[:8]:

if char != " ":

break

cursor.deleteChar()

def dedent(self):

cursor = self.text.textCursor()

if cursor.hasSelection():

# Store the current line/block number

temp = cursor.blockNumber()

# Move to the selection's last line

cursor.setPosition(cursor.anchor())

# Calculate range of selection

diff = cursor.blockNumber() - temp

direction = QtGui.QTextCursor.Up if diff > 0 else QtGui.QTextCursor.Down

# Iterate over lines

for n in range(abs(diff) + 1):

self.handleDedent(cursor)

# Move up

cursor.movePosition(direction)

else:

self.handleDedent(cursor)

def bulletList(self):

cursor = self.text.textCursor()

# Insert bulleted list

cursor.insertList(QtGui.QTextListFormat.ListDisc)

def numberList(self):

cursor = self.text.textCursor()

# Insert list with numbers

cursor.insertList(QtGui.QTextListFormat.ListDecimal)

def speechRecog(self):

r = sr.Recognizer()

cursor = self.text.textCursor()

with sr.Microphone() as source:

audio = r.listen(source)

try:

text = r.recognize\_google(audio)

cursor.insertText(text)

except:

print('Sorry could not recognize your voice!')

def main():

app = QtWidgets.QApplication(sys.argv)

main = Main()

main.show()

sys.exit(app.exec\_())

if \_\_name\_\_ == "\_\_main\_\_":

main()

**FOR FIND.PY**

#PYQT5 PyQt4’s QtGui module has been split into PyQt5’s QtGui, QtPrintSupport and QtWidgets modules

from PyQt5 import QtWidgets

#PYQT5 QTextEdit, QDialog, QPushButton, QRadioButton, QGridLayout

from PyQt5 import QtGui, QtCore

from PyQt5.QtCore import Qt

import re

class Find(QtWidgets.QDialog):

def \_\_init\_\_(self, parent = None):

QtWidgets.QDialog.\_\_init\_\_(self, parent)

self.parent = parent

self.lastStart = 0

self.initUI()

def initUI(self):

# Button to search the document for something

findButton = QtWidgets.QPushButton("Find",self)

findButton.clicked.connect(self.find)

# Button to replace the last finding

replaceButton = QtWidgets.QPushButton("Replace",self)

replaceButton.clicked.connect(self.replace)

# Button to remove all findings

allButton = QtWidgets.QPushButton("Replace all",self)

allButton.clicked.connect(self.replaceAll)

# Normal mode - radio button

self.normalRadio = QtWidgets.QRadioButton("Normal",self)

# Regular Expression Mode - radio button

regexRadio = QtWidgets.QRadioButton("RegEx",self)

# The field into which to type the query

self.findField = QtWidgets.QTextEdit(self)

self.findField.resize(250,50)

# The field into which to type the text to replace the

# queried text

self.replaceField = QtWidgets.QTextEdit(self)

self.replaceField.resize(250,50)

layout = QtWidgets.QGridLayout()

layout.addWidget(self.findField,1,0,1,4)

layout.addWidget(self.normalRadio,2,2)

layout.addWidget(regexRadio,2,3)

layout.addWidget(findButton,2,0,1,2)

layout.addWidget(self.replaceField,3,0,1,4)

layout.addWidget(replaceButton,4,0,1,2)

layout.addWidget(allButton,4,2,1,2)

self.setGeometry(300,300,360,250)

self.setWindowTitle("Find and Replace")

self.setLayout(layout)

# By default the normal mode is activated

self.normalRadio.setChecked(True)

def find(self):

# Grab the parent's text

text = self.parent.text.toPlainText()

# And the text to find

query = self.findField.toPlainText()

if self.normalRadio.isChecked():

# Use normal string search to find the query from the

# last starting position

self.lastStart = text.find(query,self.lastStart + 1)

# If the find() method didn't return -1 (not found)

if self.lastStart >= 0:

end = self.lastStart + len(query)

self.moveCursor(self.lastStart,end)

else:

# Make the next search start from the begining again

self.lastStart = 0

self.parent.text.moveCursor(QtGui.QTextCursor.End)

else:

# Compile the pattern

pattern = re.compile(query)

# The actual search

match = pattern.search(text,self.lastStart + 1)

if match:

self.lastStart = match.start()

self.moveCursor(self.lastStart,match.end())

else:

self.lastStart = 0

# We set the cursor to the end if the search was unsuccessful

self.parent.text.moveCursor(QtGui.QTextCursor.End)

def replace(self):

# Grab the text cursor

cursor = self.parent.text.textCursor()

# Security

if cursor.hasSelection():

# We insert the new text, which will override the selected

# text

cursor.insertText(self.replaceField.toPlainText())

# And set the new cursor

self.parent.text.setTextCursor(cursor)

def replaceAll(self):

self.lastStart = 0

self.find()

# Replace and find until self.lastStart is 0 again

while self.lastStart:

self.replace()

self.find()

def moveCursor(self,start,end):

# We retrieve the QTextCursor object from the parent's QTextEdit

cursor = self.parent.text.textCursor()

# Then we set the position to the beginning of the last match

cursor.setPosition(start)

# Next we move the Cursor by over the match and pass the KeepAnchor parameter

# which will make the cursor select the the match's text

cursor.movePosition(QtGui.QTextCursor.Right,QtGui.QTextCursor.KeepAnchor,end - start)

# And finally we set this new cursor as the parent's

self.parent.text.setTextCursor(cursor)

**FOR TABLE.PY**

import sys

#PYQT5 PyQt4’s QtGui module has been split into PyQt5’s QtGui, QtPrintSupport and QtWidgets modules

from PyQt5 import QtWidgets

#PYQT5 QSpinBox, QMessageBox, QDialog, QPushButton, QGridLayout, QLabel

from PyQt5 import QtGui, QtCore

from PyQt5.QtCore import Qt

class Table(QtWidgets.QDialog):

def \_\_init\_\_(self,parent = None):

QtWidgets.QDialog.\_\_init\_\_(self, parent)

self.parent = parent

self.initUI()

def initUI(self):

# Rows

rowsLabel = QtWidgets.QLabel("Rows: ",self)

self.rows = QtWidgets.QSpinBox(self)

# Columns

colsLabel = QtWidgets.QLabel("Columns",self)

self.cols = QtWidgets.QSpinBox(self)

# Cell spacing (distance between cells)

spaceLabel = QtWidgets.QLabel("Cell spacing",self)

self.space = QtWidgets.QSpinBox(self)

# Cell padding (distance between cell and inner text)

padLabel = QtWidgets.QLabel("Cell padding",self)

self.pad = QtWidgets.QSpinBox(self)

self.pad.setValue(10)

# Button

insertButton = QtWidgets.QPushButton("Insert",self)

insertButton.clicked.connect(self.insert)

# Layout

layout = QtWidgets.QGridLayout()

layout.addWidget(rowsLabel,0,0)

layout.addWidget(self.rows,0,1)

layout.addWidget(colsLabel,1,0)

layout.addWidget(self.cols,1,1)

layout.addWidget(padLabel,2,0)

layout.addWidget(self.pad,2,1)

layout.addWidget(spaceLabel,3,0)

layout.addWidget(self.space,3,1)

layout.addWidget(insertButton,4,0,1,2)

self.setWindowTitle("Insert Table")

self.setGeometry(300,300,200,100)

self.setLayout(layout)

def insert(self):

cursor = self.parent.text.textCursor()

# Get the configurations

rows = self.rows.value()

cols = self.cols.value()

if not rows or not cols:

popup = QtWidgets.QMessageBox(QtWidgets.QMessageBox.Warning,

"Parameter error",

"Row and column numbers may not be zero!",

QtWidgets.QMessageBox.Ok,

self)

popup.show()

else:

padding = self.pad.value()

space = self.space.value()

# Set the padding and spacing

fmt = QtGui.QTextTableFormat()

fmt.setCellPadding(padding)

fmt.setCellSpacing(space)

# Inser the new table

cursor.insertTable(rows,cols,fmt)

self.close()

**FOR DATETIME.PY**

#PYQT5 PyQt4’s QtGui module has been split into PyQt5’s QtGui, QtPrintSupport and QtWidgets modules

from PyQt5 import QtWidgets

#PYQT5 QDialog, QPushButton, QGridLayout, QComboBox

from PyQt5 import QtGui, QtCore

from PyQt5.QtCore import Qt

from time import strftime

class DateTime(QtWidgets.QDialog):

def \_\_init\_\_(self,parent = None):

QtWidgets.QDialog.\_\_init\_\_(self, parent)

self.parent = parent

self.formats = ["%A, %d. %B %Y %H:%M",

"%A, %d. %B %Y",

"%d. %B %Y %H:%M",

"%d.%m.%Y %H:%M",

"%d. %B %Y",

"%d %m %Y",

"%d.%m.%Y",

"%x",

"%X",

"%H:%M"]

self.initUI()

def initUI(self):

self.box = QtWidgets.QComboBox(self)

for i in self.formats:

self.box.addItem(strftime(i))

insert = QtWidgets.QPushButton("Insert",self)

insert.clicked.connect(self.insert)

cancel = QtWidgets.QPushButton("Cancel",self)

cancel.clicked.connect(self.close)

layout = QtWidgets.QGridLayout()

layout.addWidget(self.box,0,0,1,2)

layout.addWidget(insert,1,0)

layout.addWidget(cancel,1,1)

self.setGeometry(300,300,400,80)

self.setWindowTitle("Date and Time")

self.setLayout(layout)

def insert(self):

# Grab cursor

cursor = self.parent.text.textCursor()

datetime = strftime(self.formats[self.box.currentIndex()])

# Insert the comboBox's current text

cursor.insertText(datetime)

# Close the window

self.close()

**FOR DATETIME.PY**

#PYQT5 PyQt4’s QtGui module has been split into PyQt5’s QtGui, QtPrintSupport and QtWidgets modules

from PyQt5 import QtWidgets

#PYQT5 QDialog, QGridLayout, QLabel, QWidget

from PyQt5 import QtGui, QtCore

from PyQt5.QtCore import Qt

class WordCount(QtWidgets.QDialog):

def \_\_init\_\_(self,parent = None):

QtWidgets.QDialog.\_\_init\_\_(self, parent)

self.parent = parent

self.initUI()

def initUI(self):

# Word count in selection

currentLabel = QtWidgets.QLabel("Current selection",self)

currentLabel.setStyleSheet("font-weight:bold; font-size: 15px;")

currentWordsLabel = QtWidgets.QLabel("Words: ", self)

currentSymbolsLabel = QtWidgets.QLabel("Symbols: ",self)

self.currentWords = QtWidgets.QLabel(self)

self.currentSymbols = QtWidgets.QLabel(self)

# Total word/symbol count

totalLabel = QtWidgets.QLabel("Total",self)

totalLabel.setStyleSheet("font-weight:bold; font-size: 15px;")

totalWordsLabel = QtWidgets.QLabel("Words: ", self)

totalSymbolsLabel = QtWidgets.QLabel("Symbols: ",self)

self.totalWords = QtWidgets.QLabel(self)

self.totalSymbols = QtWidgets.QLabel(self)

# Layout

layout = QtWidgets.QGridLayout(self)

layout.addWidget(currentLabel,0,0)

layout.addWidget(currentWordsLabel,1,0)

layout.addWidget(self.currentWords,1,1)

layout.addWidget(currentSymbolsLabel,2,0)

layout.addWidget(self.currentSymbols,2,1)

spacer = QtWidgets.QWidget()

spacer.setFixedSize(0,5)

layout.addWidget(spacer,3,0)

layout.addWidget(totalLabel,4,0)

layout.addWidget(totalWordsLabel,5,0)

layout.addWidget(self.totalWords,5,1)

layout.addWidget(totalSymbolsLabel,6,0)

layout.addWidget(self.totalSymbols,6,1)

self.setWindowTitle("Word count")

self.setGeometry(300,300,200,200)

self.setLayout(layout)

def getText(self):

# Get the text currently in selection

text = self.parent.text.textCursor().selectedText()

# Split the text to get the word count

words = str(len(text.split()))

# And just get the length of the text for the symbols

# count

symbols = str(len(text))

self.currentWords.setText(words)

self.currentSymbols.setText(symbols)

# For the total count, same thing as above but for the

# total text

text = self.parent.text.toPlainText()

words = str(len(text.split()))

symbols = str(len(text))

self.totalWords.setText(words)

self.totalSymbols.setText(symbols)