Sprawozdanie

Praktyka Programowania PYTHON

Skład sekcji:

Konrad Kobielus

Jakub Skraba

1. Zrzuty ekranu przedstawiające wykonane zadania

A screenshot of a computer

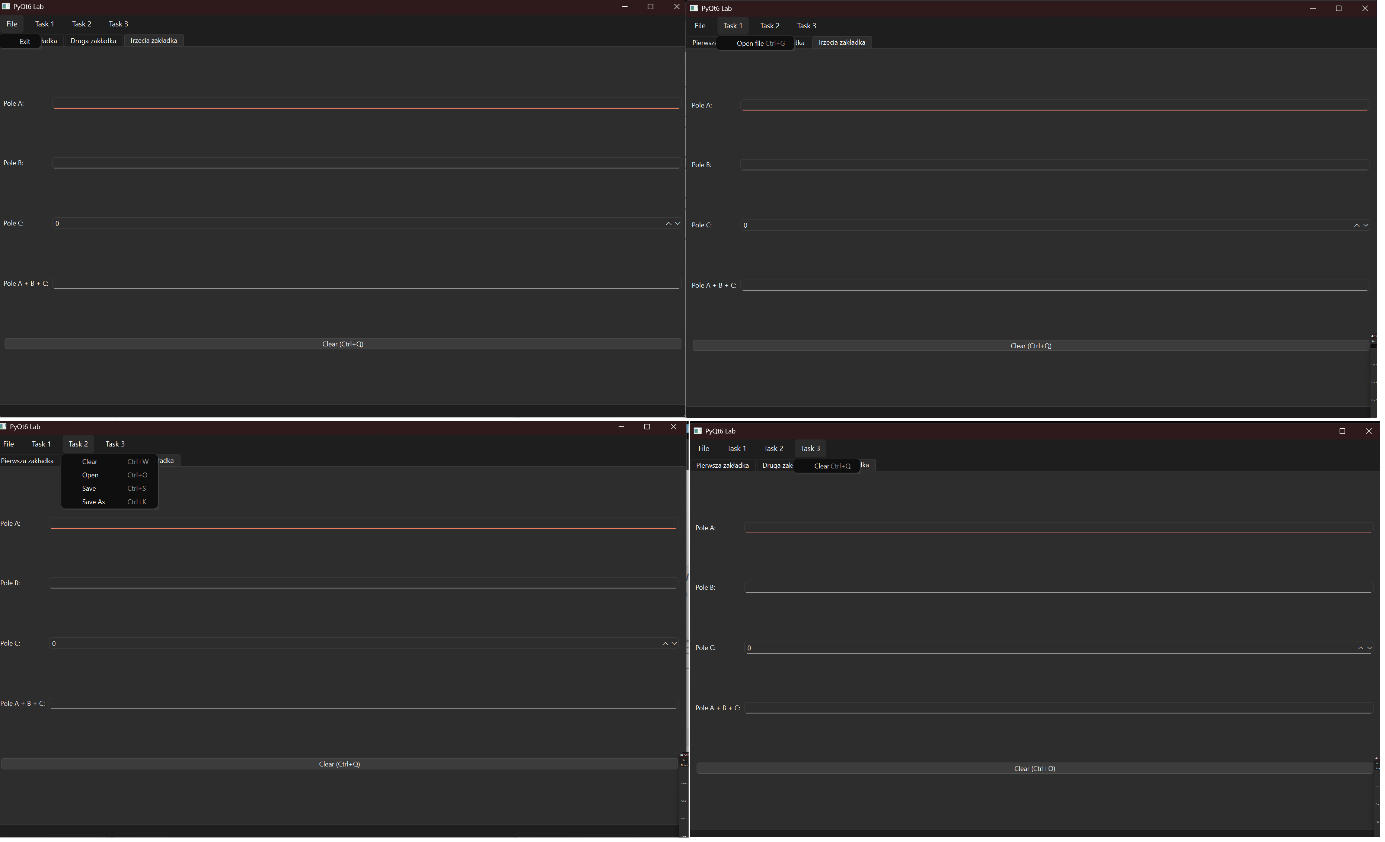
AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.



1. Kody źródłowe zadań

from PyQt6.QtWidgets import QApplication, QGridLayout, QTabWidget, QWidget, QSpinBox

from PyQt6.QtWidgets import QLabel, QLineEdit, QPushButton, QTextEdit

from PyQt6.QtWidgets import QMainWindow

from PyQt6.QtWidgets import QStatusBar

from PyQt6.QtWidgets import QToolBar

from PyQt6.QtWidgets import QFileDialog

from PyQt6.QtCore import Qt

from PyQt6.QtGui import QIcon, QAction, QPixmap, QKeySequence

# Tworzenie klasy głównego okna aplikacji dziedziczącej po QMainWindow

class Window(QMainWindow):

# Dodanie konstruktora przyjmującego okno nadrzędne

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

super().\_\_init\_\_(parent)

self.image\_path = None

self.setWindowTitle("PyQt6 Lab")

self.setGeometry(100, 100, 1240, 720)

self.createMenu()

self.createTabs()

# Funkcja dodająca pasek menu do okna

def createMenu(self):

# Stworzenie paska menu

self.menu = self.menuBar()

# Dodanie do paska listy rozwijalnej o nazwie File

self.fileMenu = self.menu.addMenu("File")

self.actionExit = QAction("Exit", self)

self.actionExit.triggered.connect(self.close)

self.fileMenu.addAction(self.actionExit)

self.task1Menu = self.menu.addMenu("Task 1")

self.actionOpen = QAction("Open file", self)

self.actionOpen.setShortcut(QKeySequence("Ctrl+G"))

self.actionOpen.triggered.connect(self.openImageDialog)

self.task1Menu.addAction(self.actionOpen)

self.task2Menu = self.menu.addMenu("Task 2")

self.actionOpen = QAction("Clear", self)

self.actionOpen.setShortcut(QKeySequence("Ctrl+W"))

self.actionOpen.triggered.connect(self.ClearTxtBox)

self.task2Menu.addAction(self.actionOpen)

self.actionOpen = QAction("Open", self)

self.actionOpen.setShortcut(QKeySequence("Ctrl+O"))

self.actionOpen.triggered.connect(self.OpenTxtFile)

self.task2Menu.addAction(self.actionOpen)

self.actionOpen = QAction("Save", self)

self.actionOpen.setShortcut(QKeySequence("Ctrl+S"))

self.actionOpen.triggered.connect(self.SaveTxtFile)

self.task2Menu.addAction(self.actionOpen)

self.actionOpen = QAction("Save As", self)

self.actionOpen.setShortcut(QKeySequence("Ctrl+K"))

self.actionOpen.triggered.connect(self.SaveAsTxtFile)

self.task2Menu.addAction(self.actionOpen)

self.task3Menu = self.menu.addMenu("Task 3")

self.actionClearTab3 = QAction("Clear", self)

self.actionClearTab3.setShortcut(QKeySequence("Ctrl+Q"))

self.actionClearTab3.triggered.connect(self.clearTab3Fields)

self.task3Menu.addAction(self.actionClearTab3)

# Funkcja dodająca wenętrzeny widżet do okna

def createTabs(self):

# Tworzenie widżetu posiadającego zakładki

self.tabs = QTabWidget()

# Stworzenie osobnych widżetów dla zakładek

self.tab\_1 = QWidget()

self.tab\_2 = QWidget()

self.tab\_3 = QWidget()

# Dodanie zakładek do widżetu obsługującego zakładki

self.tabs.addTab(self.tab\_1, "Pierwsza zakładka")

self.tabs.addTab(self.tab\_2, "Druga zakładka")

self.tabs.addTab(self.tab\_3, "Trzecia zakładka")

# Dodanie widżetu do głównego okna jako centralny widżet

self.setCentralWidget(self.tabs)

self.tab\_1.layout = None

self.tab\_1.layout = QGridLayout()

self.tab\_1.image\_label = QLabel("Brak obrazu")

self.tab\_1.layout.addWidget(self.tab\_1.image\_label, 0, 0)

self.tab\_1.setLayout(self.tab\_1.layout)

self.tab\_2.layout = None

self.tab\_2.layout = QGridLayout()

self.tab\_2.layout.addWidget(QLabel("Tytuł:"), 0, 0)

self.title\_field = QLineEdit()

self.tab\_2.layout.addWidget(self.title\_field, 0, 1, 1, 2)

self.tab\_2.layout.addWidget(QLabel("Treść:"), 1, 0)

self.content\_field = QTextEdit()

self.tab\_2.layout.addWidget(self.content\_field, 1, 1, 1, 2)

self.open\_button = QPushButton("Open")

self.save\_button = QPushButton("Zapisz")

self.save\_as\_button = QPushButton("Save As")

self.clear\_button = QPushButton("Wyczyść")

self.open\_button.clicked.connect(self.OpenTxtFile)

self.save\_button.clicked.connect(self.SaveTxtFile)

self.save\_as\_button.clicked.connect(self.SaveAsTxtFile)

self.clear\_button.clicked.connect(self.ClearTxtBox)

self.tab\_2.layout.addWidget(self.open\_button, 2, 0)

self.tab\_2.layout.addWidget(self.save\_button, 2, 1)

self.tab\_2.layout.addWidget(self.save\_as\_button, 2, 2)

self.tab\_2.layout.addWidget(self.clear\_button, 2, 3)

self.tab\_2.setLayout(self.tab\_2.layout)

self.tab\_3.layout = None

self.tab\_3.layout = QGridLayout()

self.tab\_3.layout.addWidget(QLabel("Pole A:"), 0, 0)

self.field\_a = QLineEdit()

self.field\_a.textChanged.connect(self.updateConcatenatedField)

self.tab\_3.layout.addWidget(self.field\_a, 0, 1)

self.tab\_3.layout.addWidget(QLabel("Pole B:"), 1, 0)

self.field\_b = QLineEdit()

self.field\_b.textChanged.connect(self.updateConcatenatedField)

self.tab\_3.layout.addWidget(self.field\_b, 1, 1)

self.tab\_3.layout.addWidget(QLabel("Pole C:"), 2, 0)

self.field\_c = QSpinBox()

self.field\_c.setValue(0)

self.field\_c.valueChanged.connect(self.updateConcatenatedField)

self.tab\_3.layout.addWidget(self.field\_c, 2, 1)

self.tab\_3.layout.addWidget(QLabel("Pole A + B + C:"), 3, 0)

self.field\_concatenated = QLineEdit()

self.field\_concatenated.setReadOnly(True) # Pole tylko do odczytu

self.tab\_3.layout.addWidget(self.field\_concatenated, 3, 1)

self.clear\_tab3\_button = QPushButton("Clear (Ctrl+Q)")

self.clear\_tab3\_button.clicked.connect(self.clearTab3Fields)

self.tab\_3.layout.addWidget(self.clear\_tab3\_button, 4, 0, 1, 2)

self.tab\_3.setLayout(self.tab\_3.layout)

# Dodanie paska stanu do okna

self.setStatusBar(QStatusBar(self))

# Funkcja obsługująca kliknięcie przycisku na pasku narzędzi

def onMyToolBarButtonClick(self):

self.statusBar().showMessage("Kliknięto przycisk na pasku narzędzi")

def openImageDialog(self):

file\_dialog = QFileDialog(self)

file\_dialog.setNameFilter("Images (\*.png \*.jpg \*.jpeg \*.bmp \*.gif)")

if file\_dialog.exec():

selected\_files = file\_dialog.selectedFiles()

if selected\_files:

self.image\_path = selected\_files[0]

self.statusBar().showMessage(f"Wybrano plik: {self.image\_path}")

self.displayImageOnTab1(self.image\_path)

def displayImageOnTab1(self, image\_path):

pixmap = QPixmap(image\_path)

if not pixmap.isNull():

self.tab\_1.image\_label.setPixmap(

pixmap.scaled(

600,

600,

Qt.AspectRatioMode.KeepAspectRatio,

Qt.TransformationMode.SmoothTransformation,

)

)

self.tab\_1.image\_label.setText(

""

) # Usunięcie tekstu jeśli obraz jest wyświetlany

else:

self.tab\_1.image\_label.setText("Nie można załadować obrazu")

def ClearTxtBox(self):

self.title\_field.clear()

self.content\_field.clear()

self.statusBar().showMessage("Wyczyszczono pola tekstowe")

def OpenTxtFile(self):

file\_dialog = QFileDialog(self)

file\_dialog.setNameFilter("Text files (\*.txt);;All files (\*.\*)")

if file\_dialog.exec():

selected\_files = file\_dialog.selectedFiles()

if selected\_files:

txt\_path = selected\_files[0]

try:

with open(txt\_path, "r", encoding="utf-8") as file:

content = file.read()

# Wstaw nazwę pliku jako tytuł

import os

filename = os.path.basename(txt\_path)

self.title\_field.setText(filename)

# Wstaw zawartość pliku do pola tekstowego

self.content\_field.setPlainText(content)

self.statusBar().showMessage(f"Otwarto plik: {txt\_path}")

except Exception as e:

self.statusBar().showMessage(

f"Błąd podczas otwierania pliku: {str(e)}"

)

def SaveTxtFile(self):

title = self.title\_field.text()

content = self.content\_field.toPlainText()

if not title and not content:

self.statusBar().showMessage("Brak treści do zapisania")

return

if not title:

title = "untitled.txt"

if not title.endswith(".txt"):

title += ".txt"

try:

with open(title, "w", encoding="utf-8") as file:

file.write(content)

self.statusBar().showMessage(f"Zapisano plik: {title}")

except Exception as e:

self.statusBar().showMessage(f"Błąd podczas zapisywania: {str(e)}")

def SaveAsTxtFile(self):

title = self.title\_field.text()

content = self.content\_field.toPlainText()

if not title and not content:

self.statusBar().showMessage("Brak treści do zapisania")

return

file\_dialog = QFileDialog(self)

file\_dialog.setAcceptMode(QFileDialog.AcceptMode.AcceptSave)

file\_dialog.setNameFilter("Text files (\*.txt);;All files (\*.\*)")

file\_dialog.setDefaultSuffix("txt")

if title:

if title.endswith(".txt"):

title = title[:-4]

file\_dialog.selectFile(title)

if file\_dialog.exec():

selected\_files = file\_dialog.selectedFiles()

if selected\_files:

file\_path = selected\_files[0]

try:

with open(file\_path, "w", encoding="utf-8") as file:

file.write(content)

import os

filename = os.path.basename(file\_path)

self.title\_field.setText(filename)

self.statusBar().showMessage(f"Zapisano plik jako: {file\_path}")

except Exception as e:

self.statusBar().showMessage(f"Błąd podczas zapisywania: {str(e)}")

def ClearTxtBox(self):

self.title\_field.clear()

self.content\_field.clear()

self.statusBar().showMessage("Wyczyszczono pola tekstowe")

def OpenTxtFile(self):

file\_dialog = QFileDialog(self)

file\_dialog.setNameFilter("Text files (\*.txt);;All files (\*.\*)")

if file\_dialog.exec():

selected\_files = file\_dialog.selectedFiles()

if selected\_files:

txt\_path = selected\_files[0]

try:

with open(txt\_path, "r", encoding="utf-8") as file:

content = file.read()

import os

filename = os.path.basename(txt\_path)

self.title\_field.setText(filename)

self.content\_field.setPlainText(content)

self.statusBar().showMessage(f"Otwarto plik: {txt\_path}")

except Exception as e:

self.statusBar().showMessage(

f"Błąd podczas otwierania pliku: {str(e)}"

)

def SaveTxtFile(self):

title = self.title\_field.text()

content = self.content\_field.toPlainText()

if not title and not content:

self.statusBar().showMessage("Brak treści do zapisania")

return

if not title:

title = "untitled.txt"

if not title.endswith(".txt"):

title += ".txt"

try:

with open(title, "w", encoding="utf-8") as file:

file.write(content)

self.statusBar().showMessage(f"Zapisano plik: {title}")

except Exception as e:

self.statusBar().showMessage(f"Błąd podczas zapisywania: {str(e)}")

def SaveAsTxtFile(self):

title = self.title\_field.text()

content = self.content\_field.toPlainText()

if not title and not content:

self.statusBar().showMessage("Brak treści do zapisania")

return

file\_dialog = QFileDialog(self)

file\_dialog.setAcceptMode(QFileDialog.AcceptMode.AcceptSave)

file\_dialog.setNameFilter("Text files (\*.txt);;All files (\*.\*)")

file\_dialog.setDefaultSuffix("txt")

if title:

if title.endswith(".txt"):

title = title[:-4]

file\_dialog.selectFile(title)

if file\_dialog.exec():

selected\_files = file\_dialog.selectedFiles()

if selected\_files:

file\_path = selected\_files[0]

try:

with open(file\_path, "w", encoding="utf-8") as file:

file.write(content)

import os

filename = os.path.basename(file\_path)

self.title\_field.setText(filename)

self.statusBar().showMessage(f"Zapisano plik jako: {file\_path}")

except Exception as e:

self.statusBar().showMessage(f"Błąd podczas zapisywania: {str(e)}")

def validateNumericInput(self):

"""Walidacja pola numerycznego C"""

text = self.field\_c.text()

if text and not text.replace(".", "").replace("-", "").isdigit():

self.field\_c.setText(text[:-1])

def updateConcatenatedField(self):

"""Aktualizuje pole łączące A + B + C"""

field\_a\_text = self.field\_a.text()

field\_b\_text = self.field\_b.text()

field\_c\_text = self.field\_c.text()

concatenated = field\_a\_text + field\_b\_text + field\_c\_text

self.field\_concatenated.setText(concatenated)

def clearTab3Fields(self):

"""Czyści wszystkie pola w zakładce 3"""

self.field\_a.clear()

self.field\_b.clear()

self.field\_c.clear()

self.field\_concatenated.clear()

self.statusBar().showMessage("Wyczyszczono pola zakładki 3")

# Uruchomienie okna

app = QApplication([])

win = Window()

win.show()

app.exec()