# **ФГБОУ ВО «Московский политехнический университет»**

## **Лабораторная работа № 7**

GUI, классы, модуль Tkinter

**Задание 1 Вариант №11**

По дисциплине:

Программирование

Выполнил

студент 1 курса

группы 201-321

Колобов К.В.

Проверил:

\_\_\_\_\_\_\_\_\_\_\_\_ Никишина И.Н.

# **МОСКВА 2020**

**Цель работы**

Получить практический навык в написании программ, использующих графический

интерфейс пользователя. Ознакомиться с такими понятиями, как класс, виджеты, массивы записей, а также с атрибутами класса и методами передачи данных между классами.

**Постановка задачи**

1. Описать запись Route, содержащую следующие поля:

* номер маршрута;
* название начального пункта;
* название конечного пункта.

1. Написать программу, выполняющую следующие действия:

* ввод данных с клавиатуры в массив, состоящий из восьми элементов типа Route; записи должны быть упорядочены по номерам маршрутов;
* вывод на экран информации о маршрутах, которые начинаются или кончаются в пункте, название которого введено с клавиатуры; если таких маршрутов нет, вывести на экран соответствующее сообщение;
* запись массива в файл под заданным с клавиатуры именем.

**Теоретическая часть**

Для выполнения поставленной задачи используются: графический модуль PyQt5, модуль для работы с форматом CSV.

**Структура**

**Класс Route**

Описывает один маршрут, хранит номер маршрута, названия начального и

конечного пунктов.

**Класс Routes**

Описывает список маршрутов. Выполняет: поиск, вывод в CSV файл, сохраняет

структуру списка маршрутов (то есть все маршруты отсортированы по возрастанию их номеров).

**Класс MainWindow**

Обеспечивает работу главного окна. Здесь же представлен вывод маршрутов по

мере их ввода, кнопки «Найти» и «Сохранить как», вызывающие окна поиска и окно вывода списка маршрутов в файл соответственно.

**Класс InputWindow**

Обеспечивает ввод маршрутов пользователем, контролируя при этом корректность

данных: не допускается ввод пустых полей, нецифровые символы в строке ввода номера маршрута. Пополняет список маршрутов. Окно ввода нельзя закрыть, пока длина списка маршрутов не равна восьми.

**Класс SearchWindow**

Выполняет поиск по списку маршрутов. Принимает на вход название начального

или конечного пункта маршрута, выводит «Маршруты, начинающиеся в пункте», «Маршруты, заканчивающиеся в пункте» или «Маршруты не найдены».

**Класс SaveWindow**

Обеспечивает ввод имени файла и вывод в него списка маршрутов. Если

переданное имя файла имеет расширение не `.csv`, то автоматически добавит его. Если имя файла не введено, но кнопка «Сохранить» нажата, то список маршрутов выводится в стандартный файл «default.csv».

**Класс ErrorWindow**

Обеспечивает вывод сообщений об ошибках в отдельном окне. Например,

сообщение выводится, если пользователь нажимает «Сохранить как» или «Найти» на главном окне, при этом не завершив ввод (то есть пока длина списка маршрутов меньше восьми).

**Описание программы**

Программа написана на Python 3.8, реализована в среде ОС GNU/Linux Ubuntu 20.04.

**Описание алгоритма**

1. Получить от пользователя восемь маршрутов.
2. Вывести их в текстовое поле в главном окне в порядке возрастания номеров.
3. Ожидать от пользователя некоторые события: нажатие на какую-либо кнопку (базовый принцип событийно-ориентированного программирования).

**Описание входных и выходных данных**

Пользователь вводит целые числа и строки, вывод осуществляется в GUI, а также в

стандартный файловый поток.

**Листинг программы**

import csv

import sys

from pathlib import Path

from typing import List, Any

import PyQt5.QtWidgets as Widgets

from PyQt5 import uic, QtCore

MAIN\_WINDOW\_PATH = Path('./ui/MainWindow.ui')

ERROR\_WINDOW\_PATH = Path('./ui/ErrorWindow.ui')

INPUT\_WINDOW\_PATH = Path('./ui/InputWindow.ui')

SAVE\_WINDOW\_PATH = Path('./ui/SaveWindow.ui')

SEARCH\_WINDOW\_PATH = Path('./ui/SearchWindow.ui')

ROUTES\_COUNT = 8

def sltext(text: str) -> str:

return text.lower().strip()

def stext(text: str) -> str:

return text.strip()

class Route:

def \_\_init\_\_(self,

num: int,

start: str,

dest: str) -> None:

self.\_num = num

self.\_start = start

self.\_dest = dest

@property

def num(self) -> int:

return self.\_num

@property

def start(self) -> str:

return self.\_start

@property

def dest(self) -> str:

return self.\_dest

def \_\_str\_\_(self) -> str:

num = f"Маршрут №{self.num}"

start = f"Из: '{self.start}'"

dest = f"В: '{self.dest}'"

return f"{num}\n{start}\n{dest}"

class Routes:

# for csv writing

DELIMITER = '\t'

QUOTECHAR = '"'

COLUMNS = ["Номер маршрута", "Начало маршрута", "Конец маршрута"]

def \_\_init\_\_(self,

routes: List[Route] = None) -> None:

self.\_routes = routes or []

def starts\_in\_point(self, point: str) -> Any:

point = sltext(point)

starts = [

route for route in self.\_routes

if sltext(route.start) == point

]

return Routes(starts)

def ends\_in\_point(self, point: str) -> Any:

point = sltext(point)

ends = [

route for route in self.\_routes

if sltext(route.dest) == point

]

return Routes(ends)

def dump(self, path: Path) -> None:

with path.open('w', encoding='utf-8', newline='') as f:

dm = self.DELIMITER

qch = self.QUOTECHAR

writer = csv.writer(

f, delimiter=dm, quotechar=qch, quoting=csv.QUOTE\_MINIMAL)

writer.writerow(self.COLUMNS)

for route in self.\_routes:

writer.writerow([route.num, route.start, route.dest])

def add(self, route: Route) -> None:

self.\_routes += [route]

self.\_routes.sort(key=lambda route: route.num)

def \_\_iter\_\_(self) -> iter:

return iter(self.\_routes)

def \_\_bool\_\_(self) -> bool:

return bool(self.\_routes)

def \_\_len\_\_(self) -> int:

return len(self.\_routes)

def \_\_iadd\_\_(self, route: Route) -> Any:

self.add(route)

return self

def \_\_str\_\_(self) -> str:

if len(self) == 0:

return ''

corner = '-' \* 25

inside = '\n' + '-' \* 20 + '\n'

res = inside.join(str(route) for route in self.\_routes)

return f"{corner}\n{res}\n{corner}"

routes = Routes()

class MainWindow(Widgets.QMainWindow):

def \_\_init\_\_(self, \*args) -> None:

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

uic.loadUi(MAIN\_WINDOW\_PATH, self)

self.initUI()

def initUI(self) -> None:

self.InputWindow = InputWindow(self, [])

self.ErrorWindow = ErrorWindow(self, [])

self.SaveWindow = SaveWindow(self, [])

self.SearchWindow = SearchWindow(self, [])

self.setWindowTitle("Маршруты")

self.SearchButton.clicked.connect(self.search)

self.DumpButton.clicked.connect(self.dump)

self.ExitButton.clicked.connect(self.close)

self.checkThreadTimer = QtCore.QTimer(self)

self.checkThreadTimer.setInterval(500)

self.checkThreadTimer.start()

self.checkThreadTimer.timeout.connect(self.show)

def show(self) -> None:

self.RoutesBrowser.setText(str(routes))

super().show()

if len(routes) != ROUTES\_COUNT:

self.InputWindow.show()

def dump(self) -> None:

if len(routes) != ROUTES\_COUNT:

self.error("Ввод маршрутов ещё не закончен!")

return

self.SaveWindow.show()

def search(self) -> None:

if len(routes) != ROUTES\_COUNT:

self.error("Ввод маршрутов ещё не закончен!")

return

self.SearchWindow.show()

def error(self, msg: str) -> None:

self.ErrorWindow.close()

self.ErrorWindow.display(msg)

def close(self) -> None:

self.ErrorWindow.close()

self.InputWindow.close()

self.SaveWindow.close()

self.SearchWindow.close()

super().close()

class InputWindow(Widgets.QWidget):

def \_\_init\_\_(self, \*args) -> None:

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

uic.loadUi(INPUT\_WINDOW\_PATH, self)

self.initUI()

def initUI(self) -> None:

self.EnterButton.clicked.connect(self.input)

self.ClearButton.clicked.connect(self.clear)

self.setWindowTitle("Ввод маршрутов")

def stext(self, obj) -> str:

return obj.text().strip()

def clear(self) -> None:

self.RNumInput.clear()

self.RStartInput.clear()

self.RDestInput.clear()

def input(self) -> None:

if not (len(self.stext(self.RNumInput)) > 0 and

len(self.stext(self.RStartInput)) and

len(self.stext(self.RDestInput)) > 0):

self.clear()

return

num = self.stext(self.RNumInput)

try:

num = int(num)

except ValueError:

self.clear()

return

start = self.stext(self.RStartInput)

dest = self.stext(self.RDestInput)

routes.add(Route(num, start, dest))

self.clear()

if len(routes) == ROUTES\_COUNT:

self.close()

self.RoutesRemain.setText(

f"Осталось ввести: {ROUTES\_COUNT - len(routes)}")

def show(self) -> None:

self.RoutesRemain.setText(

f"Осталось ввести: {ROUTES\_COUNT - len(routes)}")

super().show()

class ErrorWindow(Widgets.QWidget):

def \_\_init\_\_(self, \*args) -> None:

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

uic.loadUi(ERROR\_WINDOW\_PATH, self)

self.initUI()

def initUI(self) -> None:

self.Box.clicked.connect(self.close)

self.setWindowTitle("Ошибка")

def display(self, message: str) -> None:

error = "<center><b>Ошибка!</b></center><br>"

self.ErrorBrowser.setText(f"{error}\n{message}")

self.show()

class SaveWindow(Widgets.QWidget):

DEFAULT\_FILENAME = 'default.csv'

def \_\_init\_\_(self, \*args) -> None:

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

uic.loadUi(SAVE\_WINDOW\_PATH, self)

self.initUI()

def initUI(self) -> None:

self.ExitButton.clicked.connect(self.close)

self.SaveButton.clicked.connect(self.save)

self.setWindowTitle("Сохранить как")

def path(self) -> Path:

name = stext(self.FilenameInput.text())

if len(name) == 0:

raise ValueError("Имя файла слишком короткое или пустое")

name = f"{name}{'.csv' \* (not name.endswith('.csv'))}"

return Path.cwd() / name

def save(self) -> None:

try:

path = self.path()

except ValueError:

path = Path.cwd() / self.DEFAULT\_FILENAME

routes.dump(path)

self.clear()

self.close()

def clear(self) -> None:

self.FilenameInput.clear()

def close(self) -> None:

self.clear()

super().close()

class SearchWindow(Widgets.QWidget):

def \_\_init\_\_(self, \*args) -> None:

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

uic.loadUi(SEARCH\_WINDOW\_PATH, self)

self.initUI()

def initUI(self) -> None:

self.SearchButton.clicked.connect(self.show\_results)

self.ExitButton.clicked.connect(self.close)

self.setWindowTitle("Поиск маршрутов")

def show\_results(self) -> None:

if len(self.QueryInput.text().strip()) == 0:

self.ResultsBrowser.setText(

"<center><b>Введите запрос</b></center>")

return

query = self.QueryInput.text()

self.clear()

starts = routes.starts\_in\_point(query)

ends = routes.ends\_in\_point(query)

if not (starts or ends):

self.ResultsBrowser.setText("Маршруты не найдены")

if starts:

msg = f"Маршруты, начинающиеся в '{query}': "

self.ResultsBrowser.setText(f"{msg}\n{starts}")

if ends:

last\_text = self.ResultsBrowser.toPlainText()

msg = f"Маршруты, заканчивающиеся в '{query}': "

self.ResultsBrowser.setText(f"{last\_text}\n{msg}\n{ends}")

def clear(self) -> None:

self.ResultsBrowser.clear()

self.QueryInput.clear()

def close(self) -> None:

self.clear()

super().close()

def main() -> None:

app = Widgets.QApplication(sys.argv)

RoutesWindow = MainWindow()

RoutesWindow.show()

exit(app.exec\_())

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Результат работы программы**

Окна графического интерфейса пользователя.

**Список используемой литературы**

Документация языка Python: <https://docs.python.org/3/>