



Государственное бюджетное образовательное учреждение высшего образования
Московской области

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ
имени дважды Героя Советского Союза, летчика-космонавта А.А. Леонова

Колледж космического машиностроения и технологий

ОТЧЕТ

по учебной практике УП.01.01

по профессиональному модулю

**ПМ.01. Разработка программных модулей, программного обеспечения для
компьютерных систем**

специальность 09.02.03 «Программирование в компьютерных системах»

обучающегося 3 курса группы П1-20 формы обучения очной

Демьянова Артема Александровича

Место прохождения практики

ККМТ МГОТУ

Срок прохождения практики с «20» марта 2023 г. по «26» апреля 2023 г.

Руководитель практики: преподаватель _____ В.Н. Попов

подпись

Итоговая оценка по практике _____

СОДЕРЖАНИЕ

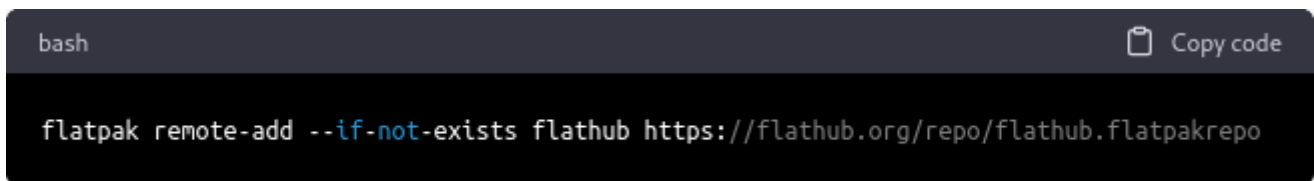
Установка и настройка среды JetBrains PyCharm.....	3
Техника работы с библиотекой tkinter.....	6
Техника работы с библиотекой NumPy.....	33
Техника работы с библиотекой Matplotlib.....	35
Техника работы с библиотекой PyQt.....	39
Техника работы с библиотекой PyGame.....	46
Техника работы с базами данных.....	51

Установка и настройка среды JetBrains PyCharm

Flatpak — технология упаковки приложений, которая позволяет устанавливать приложения, которые работают в изолированной среде на любой Linux-системе, независимо от ее версии и настроек. Приложения, упакованные с помощью Flatpak, могут включать все необходимые зависимости, чтобы работать независимо от основной части операционной системы и ее библиотек.

Установка PyCharm через Flatpak на ОС семейства Linux очень проста, быстра и позволяет легко получить последнюю версию приложения без необходимости устанавливать зависимости вручную. Кроме того, такой подход обеспечивает удобный способ обновления PyCharm до новых версий в будущем.

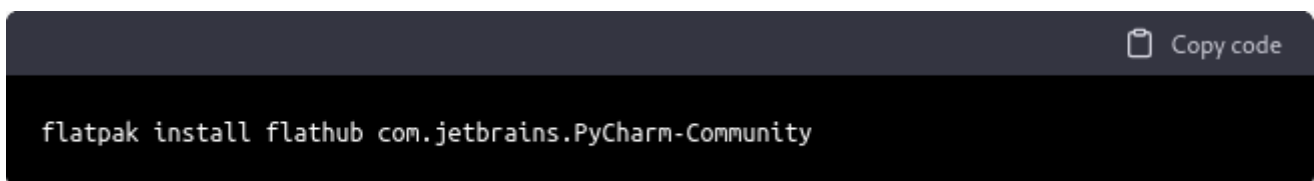
Flatpak использует репозитории, которые содержат приложения в виде пакетов. Flathub — крупнейший репозиторий приложений Flatpak, который содержит более 700 приложений, в том числе и PyCharm. Чтобы использовать Flathub, необходимо добавить его в качестве удаленного репозитория Flatpak в операционной системе. Для этого следует запустить следующую команду в терминале:

A terminal window with a dark background. The prompt is 'bash'. The command entered is 'flatpak remote-add --if-not-exists flathub https://flathub.org/repo/flathub.flatpakrepo'. There is a 'Copy code' button in the top right corner.

```
bash  
flatpak remote-add --if-not-exists flathub https://flathub.org/repo/flathub.flatpakrepo
```

Рисунок 1. Добавление репозитория Flathub

После того, как репозиторий Flathub был добавлен, нужно установить PyCharm. Для этого используется следующая команда в терминале:

A terminal window with a dark background. The command entered is 'flatpak install flathub com.jetbrains.PyCharm-Community'. There is a 'Copy code' button in the top right corner.

```
flatpak install flathub com.jetbrains.PyCharm-Community
```

Рисунок 2. Установка PyCharm

Эта команда загрузит и установит последнюю версию PyCharm Community Edition из репозитория Flathub.

После загрузки и установки, ярлык для запуска PyCharm появится в списке всех установленных программ наряду с другими установленными в системе программами:



Рисунок 3. Ярлык для запуска PyCharm

Базовая настройка среды разработки PyCharm производится в разделе «Customize» главного экрана. Здесь можно изменить цветовую тему (Color theme), размер шрифта во всей программе (IDE font):

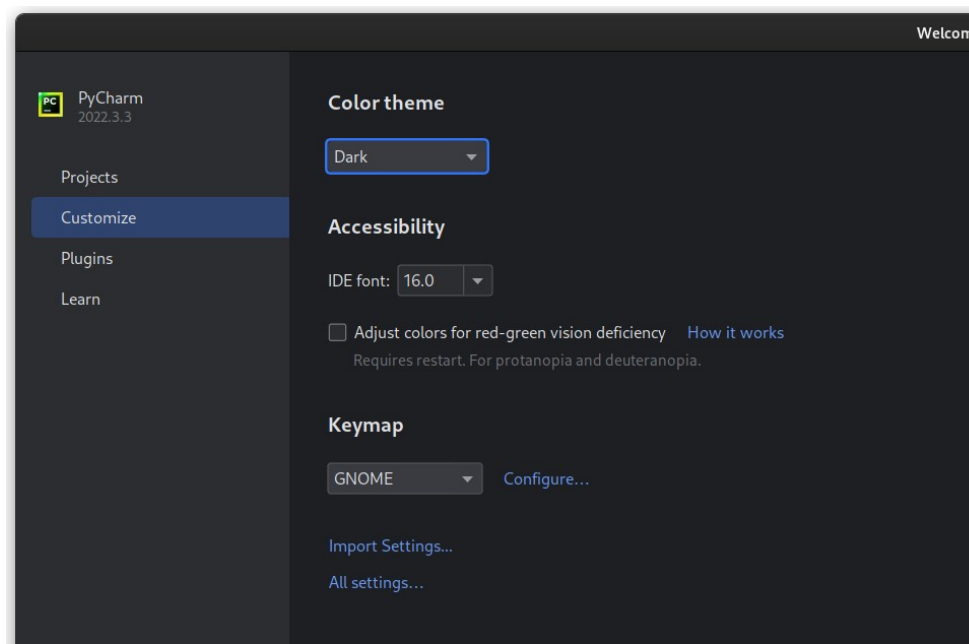


Рисунок 4. Базовая настройка среды PyCharm

Более подробная настройка производится в окне, которое вызывается по нажатию на «All settings...». В появившемся окне находятся все параметры программы. Среди их множества стоит выделить параметр для настройки директории по умолчанию для новых проектов. Изменение директории по умолчанию производится в подпункте «System Settings» пункта «Appearance & Behavior»:

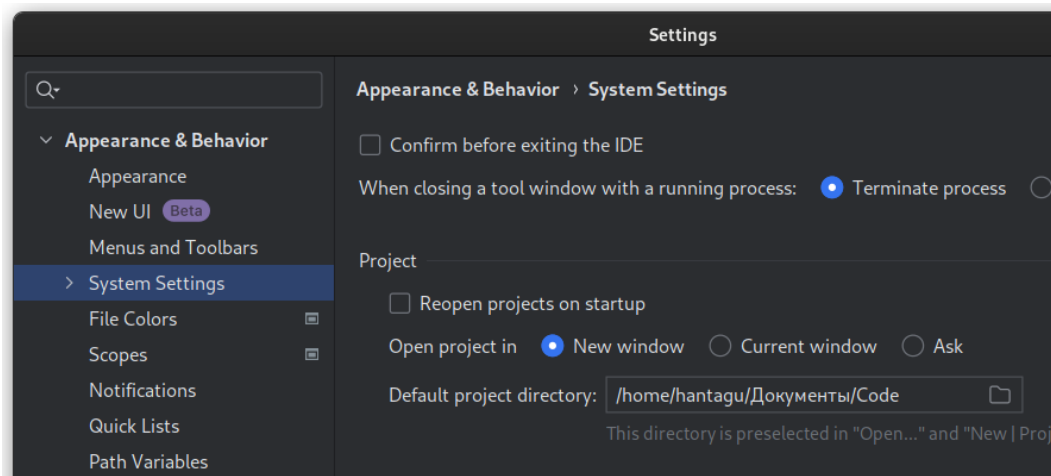


Рисунок 5. Директория по умолчанию для новых проектов

Техника работы с библиотекой tkinter

Библиотека Tkinter — стандартная библиотека Python, которая предоставляет возможности для создания графических пользовательских интерфейсов (GUI). Tkinter основана на библиотеке Tcl/Tk и содержит большое количество виджетов и методов для создания интерактивных оконных приложений.

Ниже перечислены основные функции и методы, которые можно использовать в Tkinter:

- Tk() — создает главное окно приложения;
- Label() — создает надпись на графическом интерфейсе;
- Button() — создает кнопку на графическом интерфейсе;
- Entry() — создает поле для ввода текста;
- Text() — создает многострочное поле для ввода текста;
- Frame() — создает контейнер для группировки других виджетов;
- Canvas() — создает область для рисования графики и фигур;
- Menu() — создает меню на графическом интерфейсе;
- messagebox.showinfo() — создает информационное диалоговое окно;
- messagebox.askquestion() — создает диалоговое окно с вопросом;
- messagebox.askyesno() — создает диалоговое окно с вопросом и двумя кнопками «Да» и «Нет»;
- bind() — связывает событие с функцией.

Это лишь небольшой список функций и методов, доступных в Tkinter. Более подробную информацию можно найти в документации к библиотеке. В целом, Tkinter предоставляет широкий набор инструментов для создания графических интерфейсов любой сложности и масштаба, и может быть полезна для различных проектов, которые требуют взаимодействия с пользователем.

Задание 1. Импортируйте модуль tkinter, создайте объект главного окна, примените к нему метод mainloop. Затем выполните скрипт. Что вы видите?

```
from tkinter import *  
root = Tk()  
root.mainloop()
```

При выполнении скрипта программа успешно запускается и появляется пустое окно без каких-либо виджетов:

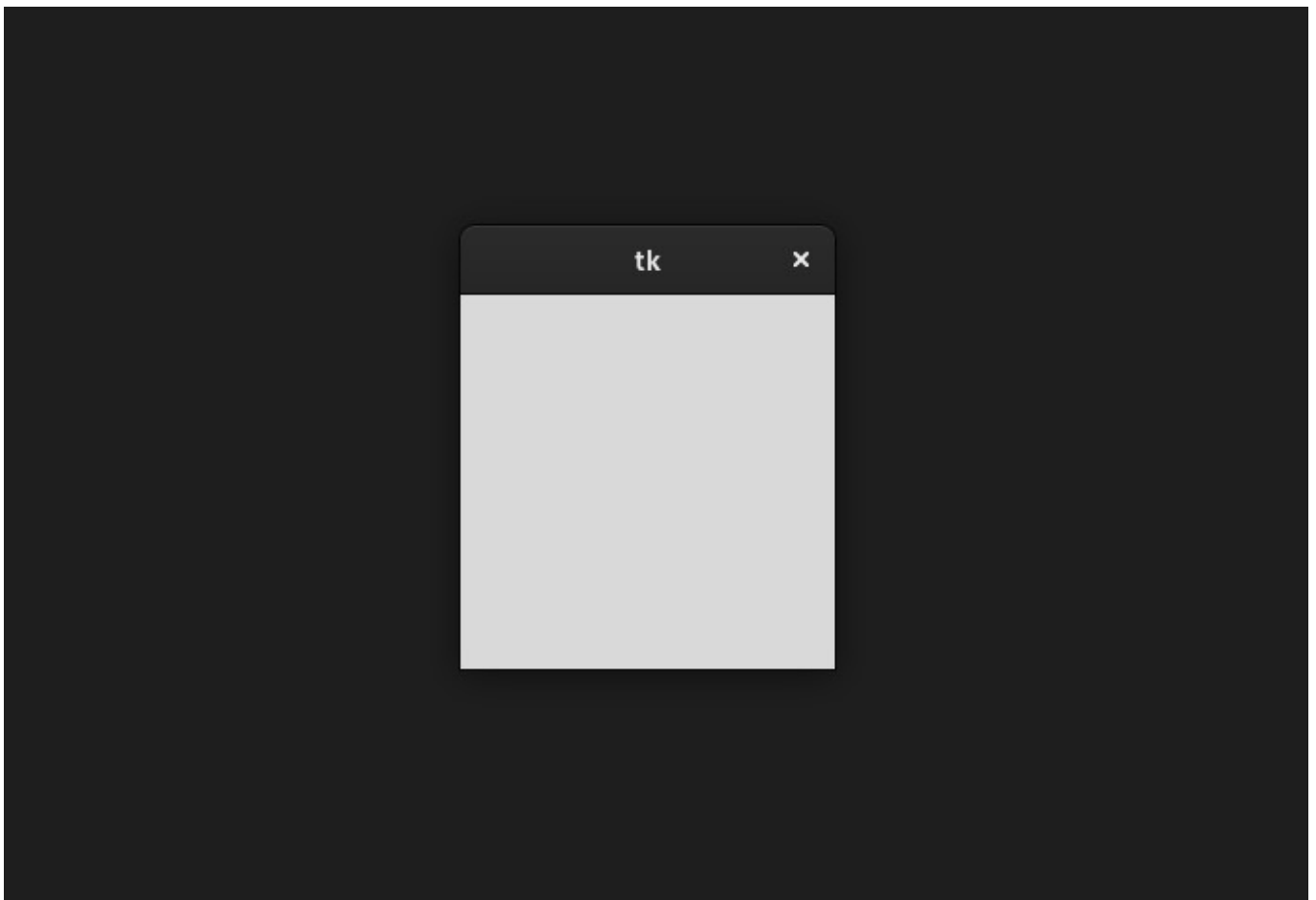


Рисунок 6. Результат выполнения Задания 1

Задание 2. Добавьте кнопку на главное окно с помощью такой команды:
`but = Button(root, text="Печать")`. В данном случае, при создании кнопки, в класс сразу передается и значение свойства `text`. Это наиболее часто используемый способ установки свойств (по сравнению с тем, который приводится в уроке: `but["text"] = "Печать"`).

```
from tkinter import *  
root = Tk()  
button = Button(root, text='Печать')  
root.mainloop()
```


Задание 3. Расположите виджету на главном окне с помощью метода `pack`. Запустите скрипт. Что вы видите? Нажмите левой кнопкой мыши на кнопку в окне. Что-нибудь происходит?

```
from tkinter import *  
root = Tk()  
button = Button(root, text='Печать')  
button.pack()  
root.mainloop()
```

При выполнении скрипта программа успешно запускается и появляется окно с единственным виджетом — кнопкой (Рисунок 7). Размер окна программы подстроился под размер кнопки. При нажатии левой кнопкой мыши по кнопке видна анимация ее нажатия, но никаких действий не происходит.

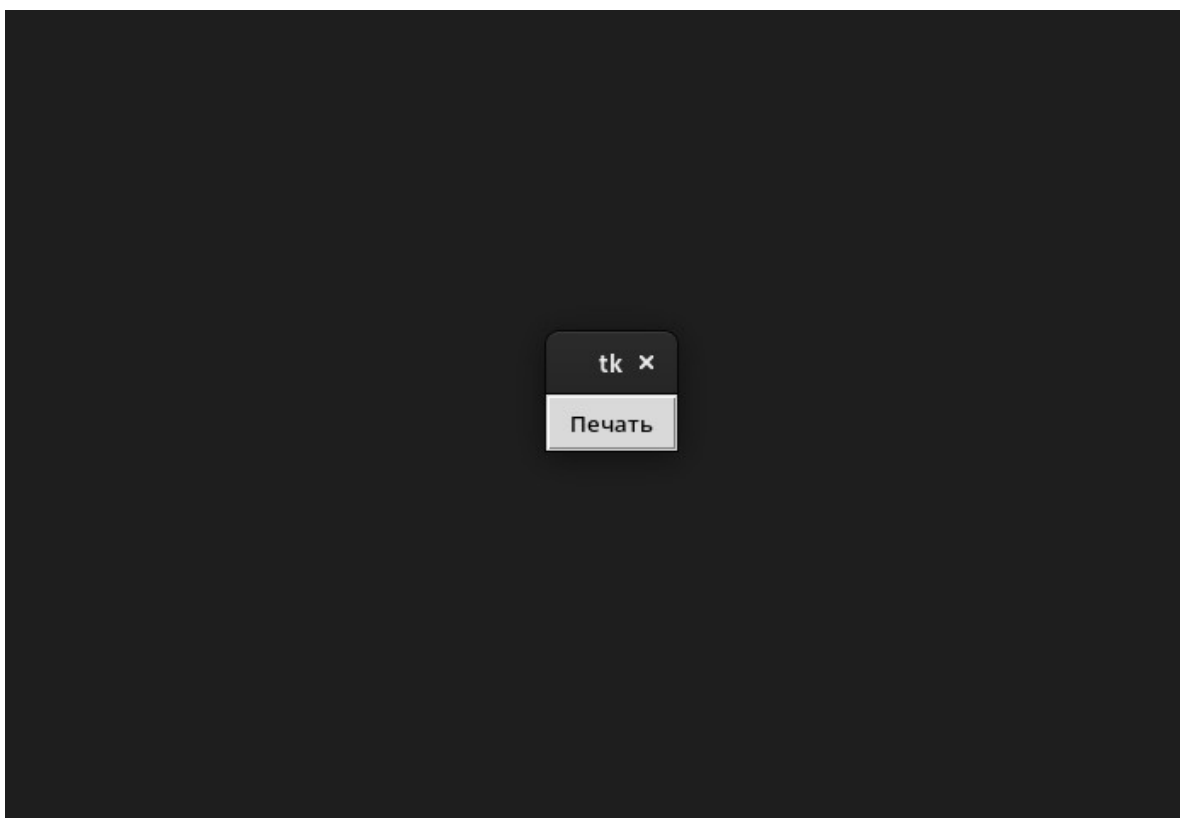


Рисунок 7. Результат выполнения Задания 3

Задание 4. Создайте какую-нибудь функцию и свяжите ее с событием нажатия кнопки. Запустите скрипт и нажмите на кнопку.

```
from tkinter import *
cnt = 0
def f(event):
    global cnt
    cnt += 1
    print(f'Количество нажатий: {cnt}')
root = Tk()
button = Button(root, text='Печать')
button.bind('<Button-1>', f)
button.pack()
root.mainloop()
```

При выполнении скрипта появляется окно с кнопкой (Рисунок 8). При нажатии левой кнопкой мыши по ней видна анимация нажатия и в терминале печатается количество нажатий на кнопку.

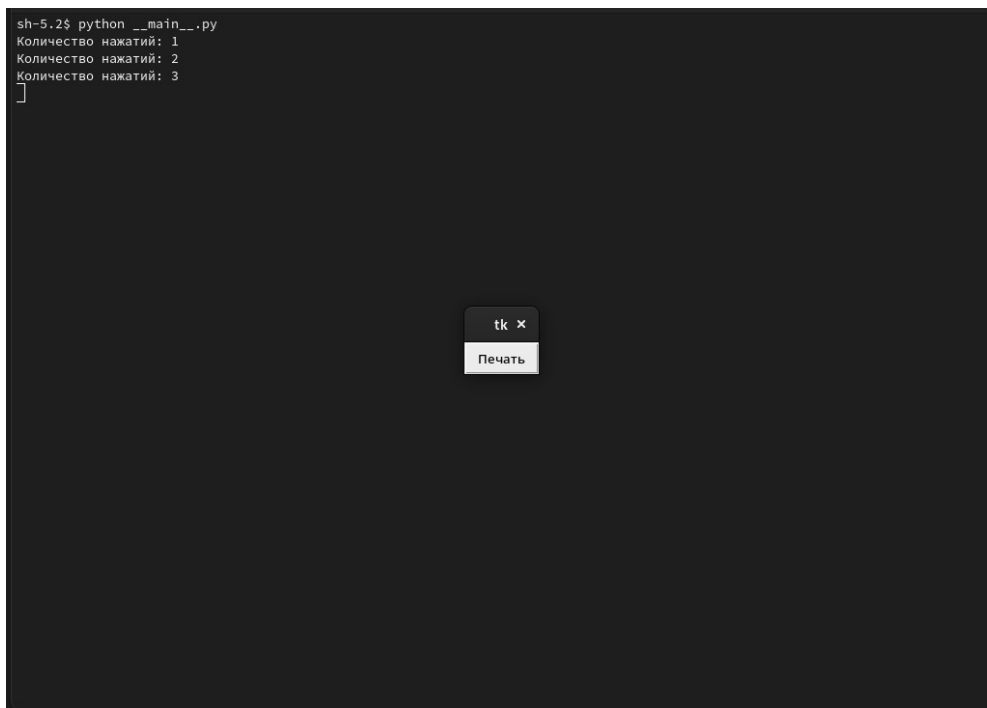


Рисунок 8. Результат выполнения Задания 4

Задание 5. Создайте два скрипта на языке программирования Python и с использованием модуля Tkinter генерирующие представленные в задании шаблоны.

Шаблон №1.

```
from tkinter import *
root = Tk()
root.configure(bg='#ece9d8')
address_label = Label(root, text='Ваш адрес:', bg='#ffffe0')
address_label.pack()
address_entry = Entry(root)
address_entry.pack()
comment_label = Label(root, text='Комментарий:', bg='#ece9d8')
comment_label.pack()
comment_text = Text(root, width=25, height=7)
comment_text.pack()
send_button = Button(root, text='Отправить', bg='#add8e6')
send_button.pack()
root.mainloop()
```

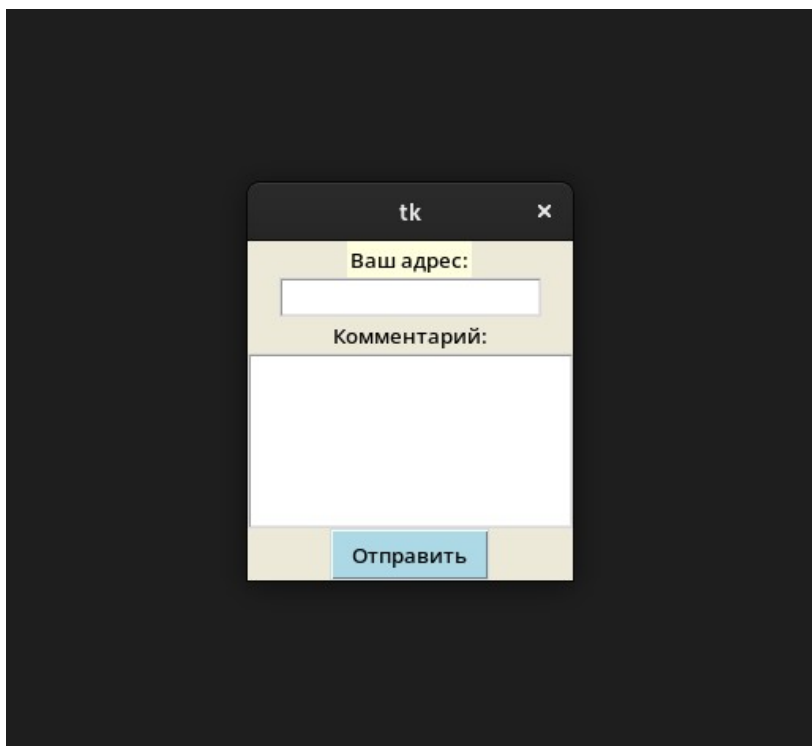


Рисунок 9. Выполненный Шаблон №1 в Задании 5

Шаблон №2.

```
from tkinter import *
root = Tk()
root.configure(bg='#ece9d8')
Label(root, text='Сколько штук?', bg='#ece9d8').pack()
amount = IntVar()
radios = []
for i, j in zip(range(10, 41, 10), range(4)):
    radios.append(Radiobutton(root, text=f'{i-9}-{i}', variable=amount, value=i))
    radios[j].configure(bg='#ece9d8')
    radios[j].pack()
Label(root, text='Какого цвета?', bg='#ece9d8').pack()
colors = (BooleanVar(), BooleanVar(), BooleanVar(), BooleanVar())
checks = []
for i, j in zip(('RED', 'BLUE', 'GREEN', 'YELLOW'), range(4)):
    checks.append(Checkbutton(root, text=i, bg=i.lower(), variable=colors[j],
onvalue=True, offvalue=False))
    checks[j].pack()
root.mainloop()
```

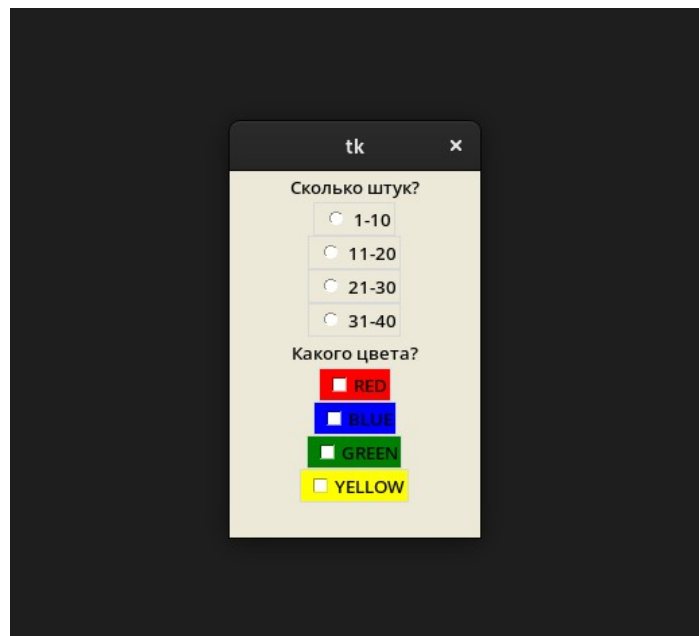


Рисунок 10. Выполненный Шаблон №2 в
Задании 5

Задание 6. Создайте два скрипта на языке программирования Python и с использованием модуля Tkinter генерирующие представленные в задании шаблоны.

Шаблон №1.

```
from tkinter import *
root = Tk()
root.configure(bg='#ece9d8')
frame1 = Frame(root, borderwidth=15, background='lime')
frame1.pack()
Scale(frame1, from_=0, to=100, tickinterval=10, resolution=10, length=250,
orient=HORIZONTAL).pack()
frame2 = Frame(root, borderwidth=15, background='green')
frame2.pack()
Entry(frame2, width=12).pack()
root.mainloop()
```

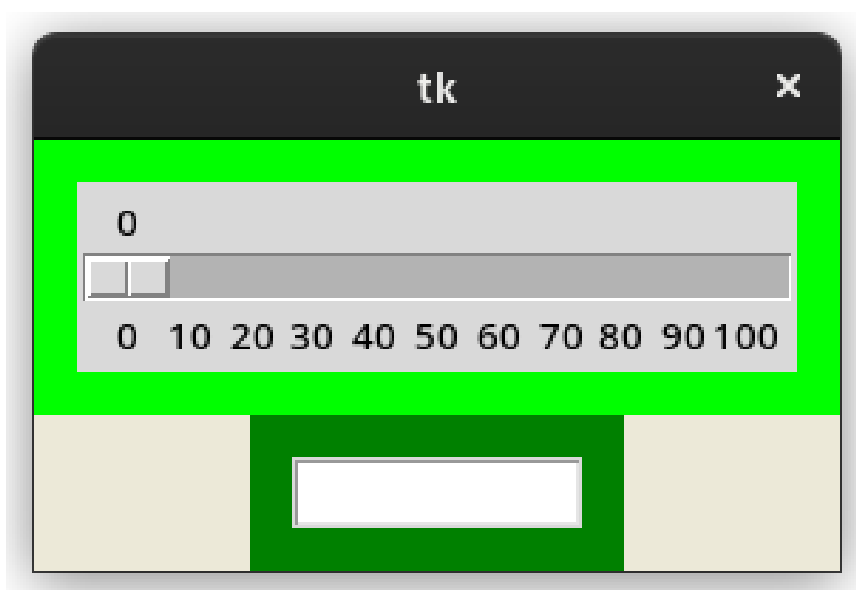


Рисунок 11. Выполненный Шаблон №1 в Задании 6

Шаблон №2.

```
from tkinter import *
root = Tk()
root.configure(bg='#ece9d8')
text = Text(root, width=30, height=5)
scrollbar = Scrollbar(root, command=text.yview)
text.configure(yscrollcommand=scrollbar.set)
text.grid(row=0, column=0)
scrollbar.grid(row=0, column=1, sticky='ns')
root.mainloop()
```

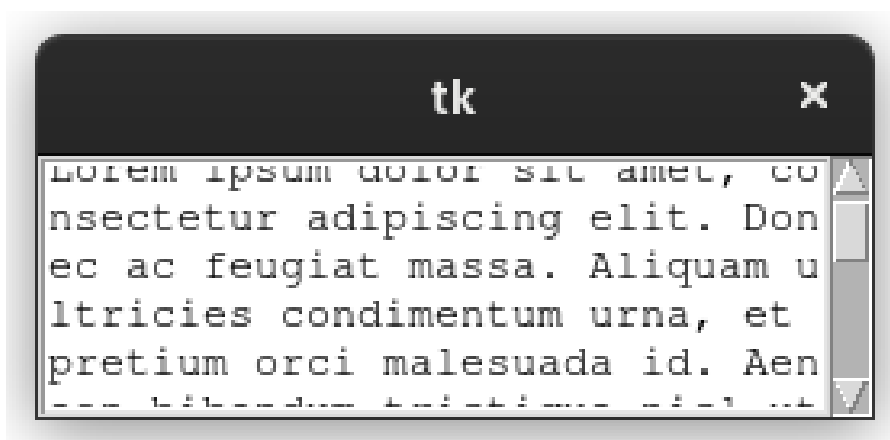


Рисунок 12. Выполненный Шаблон №2 в Задании 6

Задание 7. Создайте приложение, состоящее из главного и двух дочерних окон. На каждом из трех окон должны располагаться один или два любых графических объекта.

```
from tkinter import *  
  
def new_window(root, text):  
    window = Toplevel(root)  
    Label(window, text=text).pack()  
    button = Button(window, text='Заккрыть')  
    button.bind('<Button-1>', lambda e: window.destroy())  
    button.pack()  
  
root = Tk()  
b1 = Button(root, text='Окно 1')  
b1.bind('<Button-1>', lambda e: new_window(root, 'Окно 1'))  
b1.pack()  
b1 = Button(root, text='Окно 2')  
b1.bind('<Button-1>', lambda e: new_window(root, 'Окно 2'))  
b1.pack()  
root.mainloop()
```



Рисунок 13. Выполненное Задание 7

Задание 8. Создайте приложение, в котором меняется размер фрейма в зависимости от того, какая из трех объектов-кнопок была нажата.

```
from tkinter import *
root = Tk()
frame = Frame(root, width=50, height=50, bg='lime')
frame.pack()
inc = Button(root, text='Увеличить')
inc.bind('<Button-1>', lambda e: frame.configure(width=frame.winfo_width() + 10,
height=frame.winfo_height() + 10))
inc.pack()
dec = Button(root, text='Уменьшить')
dec.bind('<Button-1>', lambda e: frame.configure(width=frame.winfo_width() - 10,
height=frame.winfo_height() - 10))
dec.pack()
dfl = Button(root, text='Вернуть исходный размер')
dfl.bind('<Button-1>', lambda e: frame.configure(width=50, height=50))
dfl.pack()
root.mainloop()
```

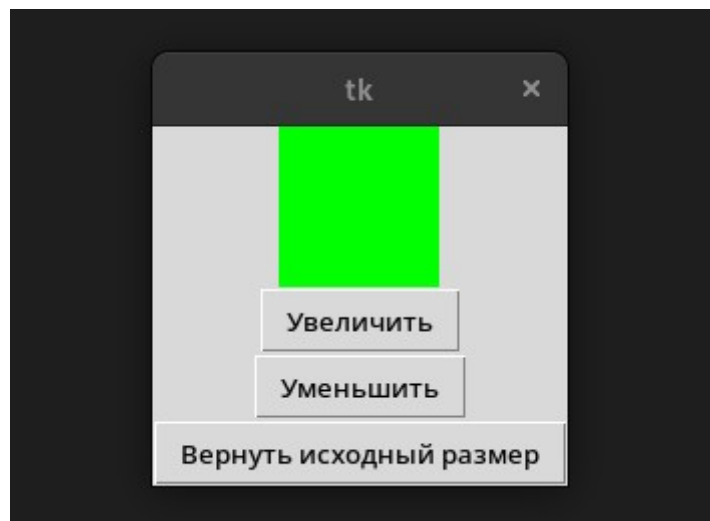


Рисунок 14. Выполненное Задание 8

Задание 9. Напишите скрипт, генерирующий окно с меткой и текстовым полем. После ввода пользователем текста в поле и нажатия Enter, введенный текст должен отображаться в метке.

```
from tkinter import *  
root = Tk()  
label = Label(text='Введите текст ниже и нажмите Enter')  
label.pack()  
entry = Entry(root)  
entry.bind('<Return>', lambda e: label.configure(text=entry.get()))  
entry.pack()  
root.mainloop()
```

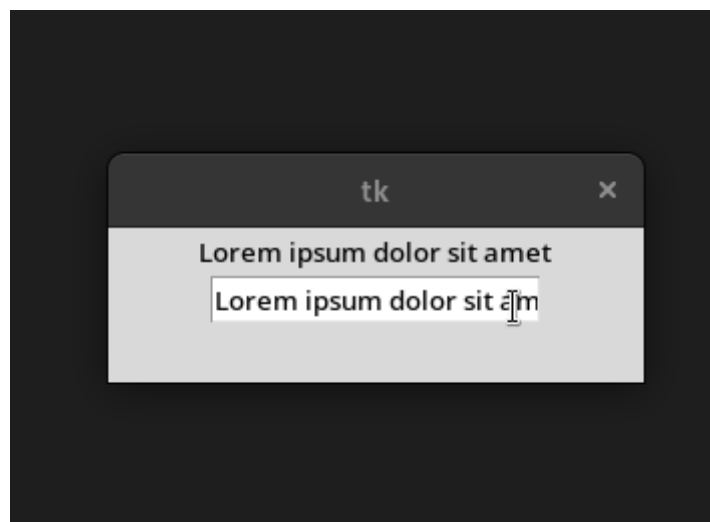


Рисунок 15. Выполненное Задание 9

Задание 10. Напишите следующую программу. На главном окне находится несколько флажков и текстовое поле. При щелчке левой кнопкой мыши в пределах текстового поля в нем должны отображаться значения включенных флажков (появляться сообщение о том, какие флажки включены), при щелчке правой кнопкой мыши — значения выключенных флажков.

```
from tkinter import *
def set_text(e: Entry, t: str):
    e.delete(0, END)
    e.insert(0, t)
root = Tk()
checks = tuple(BooleanVar() for _ in range(3))
for i in range(3):
    check = Checkbutton(root, text=f'{i}', variable=checks[i], onvalue=True,
offvalue=False)
    check.pack()
entry = Entry(root)
entry.bind('<Button-1>', lambda e: set_text(entry, ', '.join(str(k) for k, v in
enumerate(checks) if v.get()))))
entry.bind('<Button-3>', lambda e: set_text(entry, ', '.join(str(k) for k, v in
enumerate(checks) if not v.get()))))
entry.pack()
root.mainloop()
```

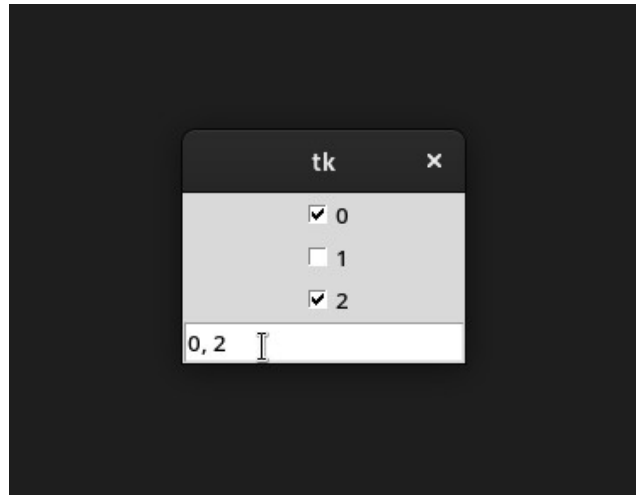


Рисунок 16. Выполненное Задание 10

Задание 11. Напишите скрипт, генерирующий в окне два текстовых поля и рамку. Размер рамки можно менять с помощью вводимых значений в текстовые поля (определяют длину и ширину) и нажатии клавиши пробел на клавиатуре.

```
from tkinter import *
root = Tk()
frame = Frame(root, width=50, height=50, background='lime')
frame.pack()
w = Entry(root)
w.pack()
h = Entry(root)
h.pack()
h.bind('<space>', lambda e: frame.configure(width = int(w.get()), height =
int(h.get()))))
w.bind('<space>', lambda e: frame.configure(width = int(w.get()), height =
int(h.get()))))
root.mainloop()
```

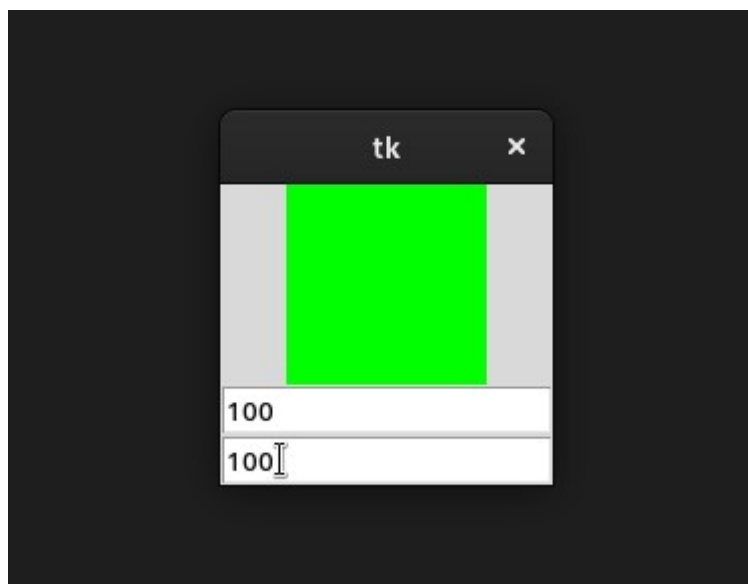


Рисунок 17. Выполненное Задание 11

Задание 12. Напишите скрипт, как в примере с флажками; в отличии от примера значения ассоциированных переменных должны отображаться в метке (Label) через запятую.

```
from tkinter import *
root = Tk()
strs = tuple(StringVar(value='-') for _ in range(3))
for i, (j, k) in enumerate(zip(('Окружность', 'Квадрат', 'Треугольник'), ('circle',
'square', 'triangle'))):
    ch = Checkbutton(root, text=j, variable=strs[i], onvalue=k, offvalue="-")
    ch.pack()
label = Label(root)
label.pack()
but = Button(root, text="Получить значения")
but.bind('<Button-1>', lambda e: label.configure(text=', '.join(i.get() for i in strs)))
but.pack()
root.mainloop()
```

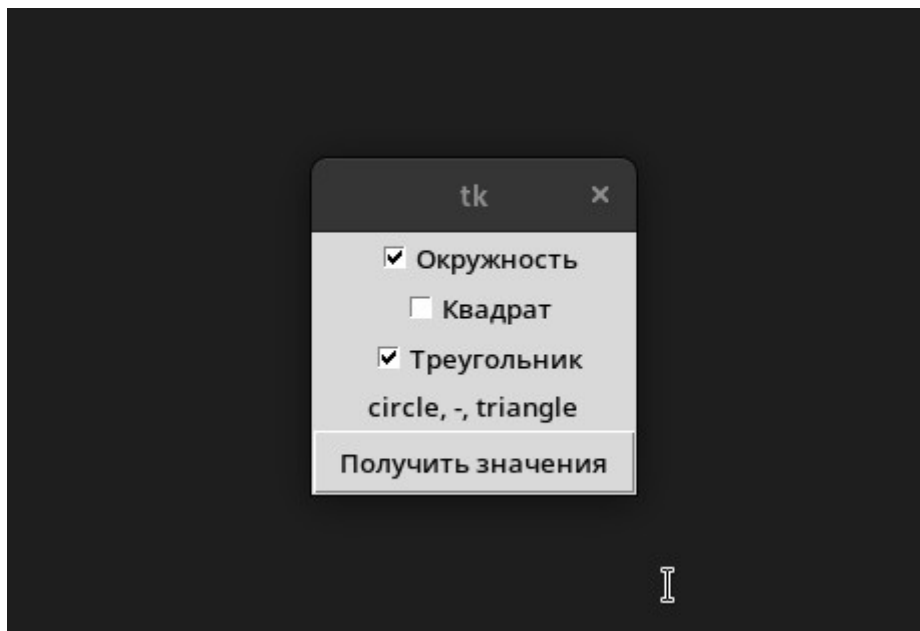


Рисунок 18. Выполненное Задание 12

Задание 13. Напишите программу, в которой пользователь может определить цвет рамки (Frame) с помощью шкалы (Scale).

```
from tkinter import *
root = Tk()
frame = Frame(root, width=75, height=75, background='black')
frame.pack()
r_wrapper = Frame(root, background='red', padx=5, pady=5)
g_wrapper = Frame(root, background='green', padx=5, pady=5)
b_wrapper = Frame(root, background='blue', padx=5, pady=5)
r = Scale(r_wrapper, from_=0, to=255, orient=HORIZONTAL, length=250)
g = Scale(g_wrapper, from_=0, to=255, orient=HORIZONTAL, length=250)
b = Scale(b_wrapper, from_=0, to=255, orient=HORIZONTAL, length=250)
button = Button(text='Установить цвет')
button.bind('<Button-1>', lambda e: frame.configure(background='#{:02x}{:02x}{:02x}'.format(int(r.get()), int(g.get()), int(b.get()))))
button.pack()
r_wrapper.pack(); g_wrapper.pack(); b_wrapper.pack()
r.pack(); g.pack(); b.pack()
root.mainloop()
```

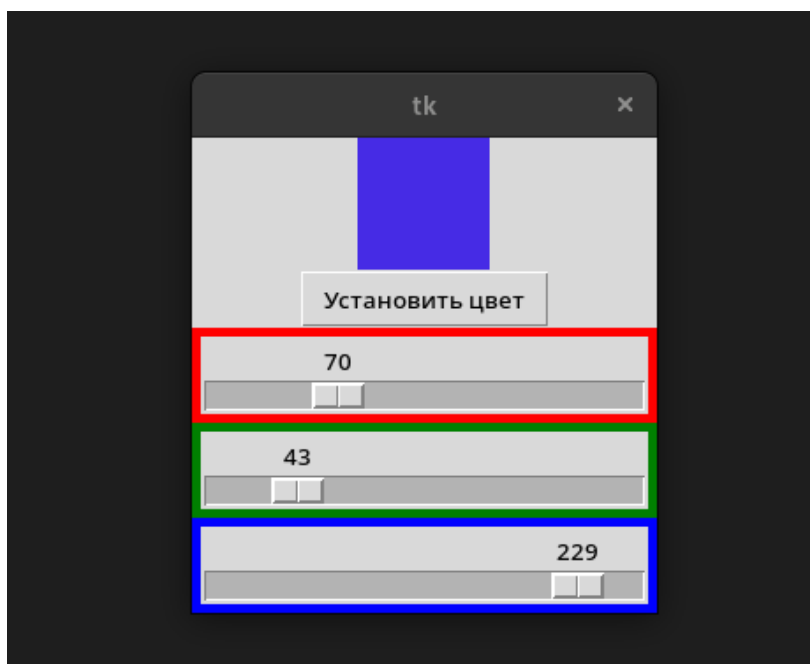


Рисунок 19. Выполненное Задание 13

Задание 14. Напишите приложение с меню, содержащим два пункта: Color и Size. Пункт Color должен содержать три команды (Red, Green и Blue), меняющие цвет рамки на главном окне. Пункт Size должен содержать две команды (500x500 и 700x400), изменяющие размер рамки.

```
from tkinter import *
def set_size(w, h):
    frame.configure(width=w, height=h)
def set_color(c):
    frame.configure(background=c)
root = Tk()
menu = Menu(root)
frame = Frame(root, background='red', width=500, height=500)
frame.pack()
color = Menu(menu)
color.add_command(label='Red', command=lambda: set_color('red'))
color.add_command(label='Green', command=lambda: set_color('green'))
color.add_command(label='Blue', command=lambda: set_color('blue'))
size = Menu(menu)
size.add_command(label='500x500', command=lambda: set_size(500, 500))
size.add_command(label='700x400', command=lambda: set_size(700, 400))
menu.add_cascade(label='Color', menu=color)
menu.add_cascade(label='Size', menu=size)
root.configure(menu=menu)
root.mainloop()
```

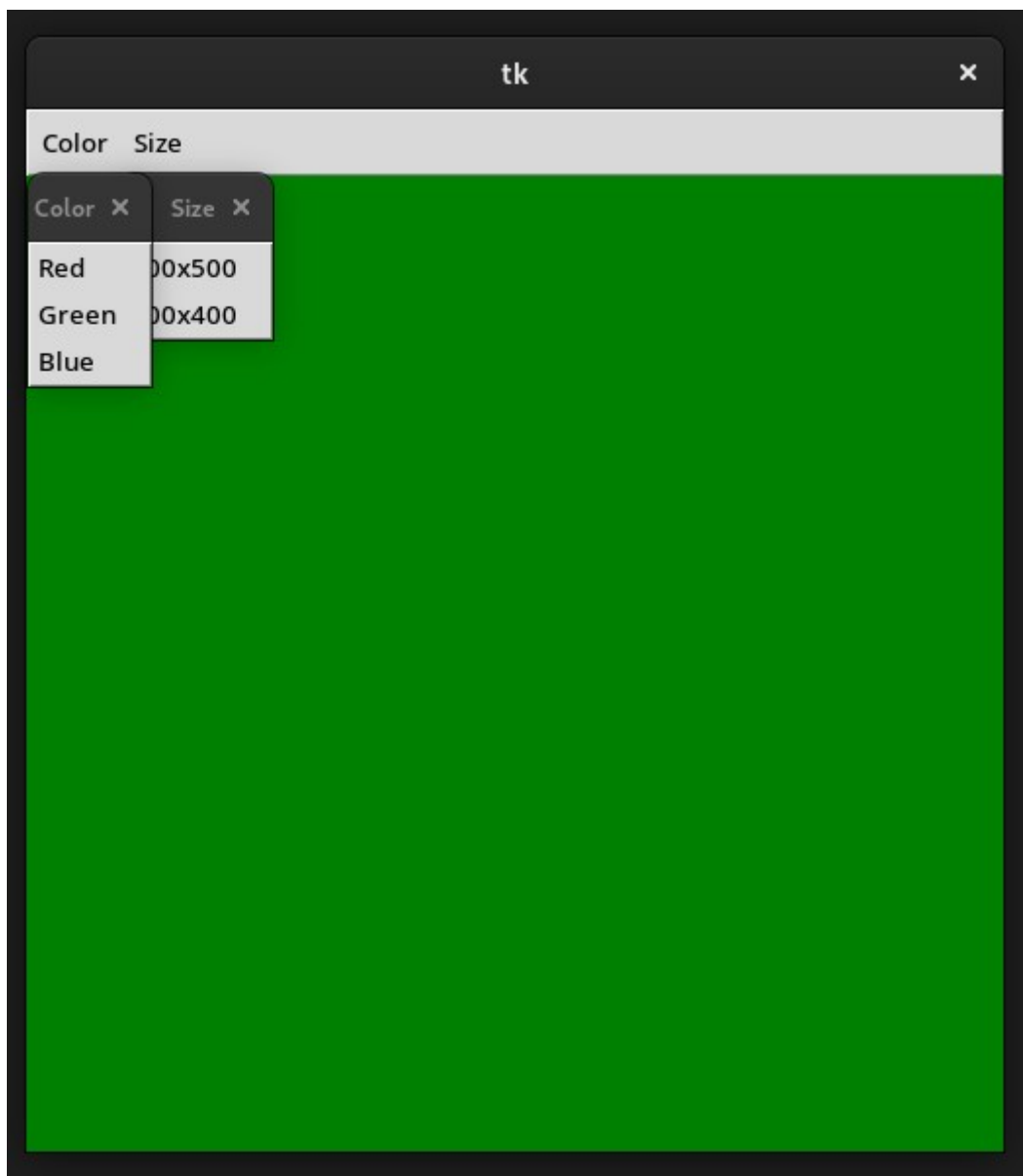


Рисунок 20. Выполненное Задание 14

Задание 15. Напишите программу, описанную в уроке. Измените программу: пусть после нажатия пункта Exit пользователю выводилось не окно с вопросом «Выйти или нет», а окно с вопросом «Сохранить или нет». В случае положительного ответа должна вызываться функция `_save` и только затем завершаться приложение. Если в текстовом поле что-то содержится, то при открытии файла оно не удаляется, а содержимое файла просто дописывается. Исправьте этот недостаток (перед открытием файла содержимое текстового поля должно удаляться).

```
from tkinter import *
from tkinter.filedialog import *
from tkinter.messagebox import *

def _open():
    path = askopenfilename()
    with open(path, 'rt') as f:
        txt.delete(0.0, END)
        txt.insert(0.0, f.read())

def _save():
    path = asksaveasfilename()
    with open(path, 'wt') as f:
        f.write(txt.get(0.0, END))
    root.destroy()

def _exit():
    if askyesno(message='Сохранить файл?'):
        _save()
    root.destroy()

root = Tk()
menu = Menu(root)
file = Menu(menu)
file.add_command(label='Открыть...', command=_open)
file.add_command(label='Сохранить как...', command=_save)
file.add_command(label='Выйти', command=_exit)
```

```
menu.add_cascade(label='Файл', menu=file)
root.configure(menu=menu)
txt = Text(root, width=40, height=15)
txt.pack()
root.mainloop()
```

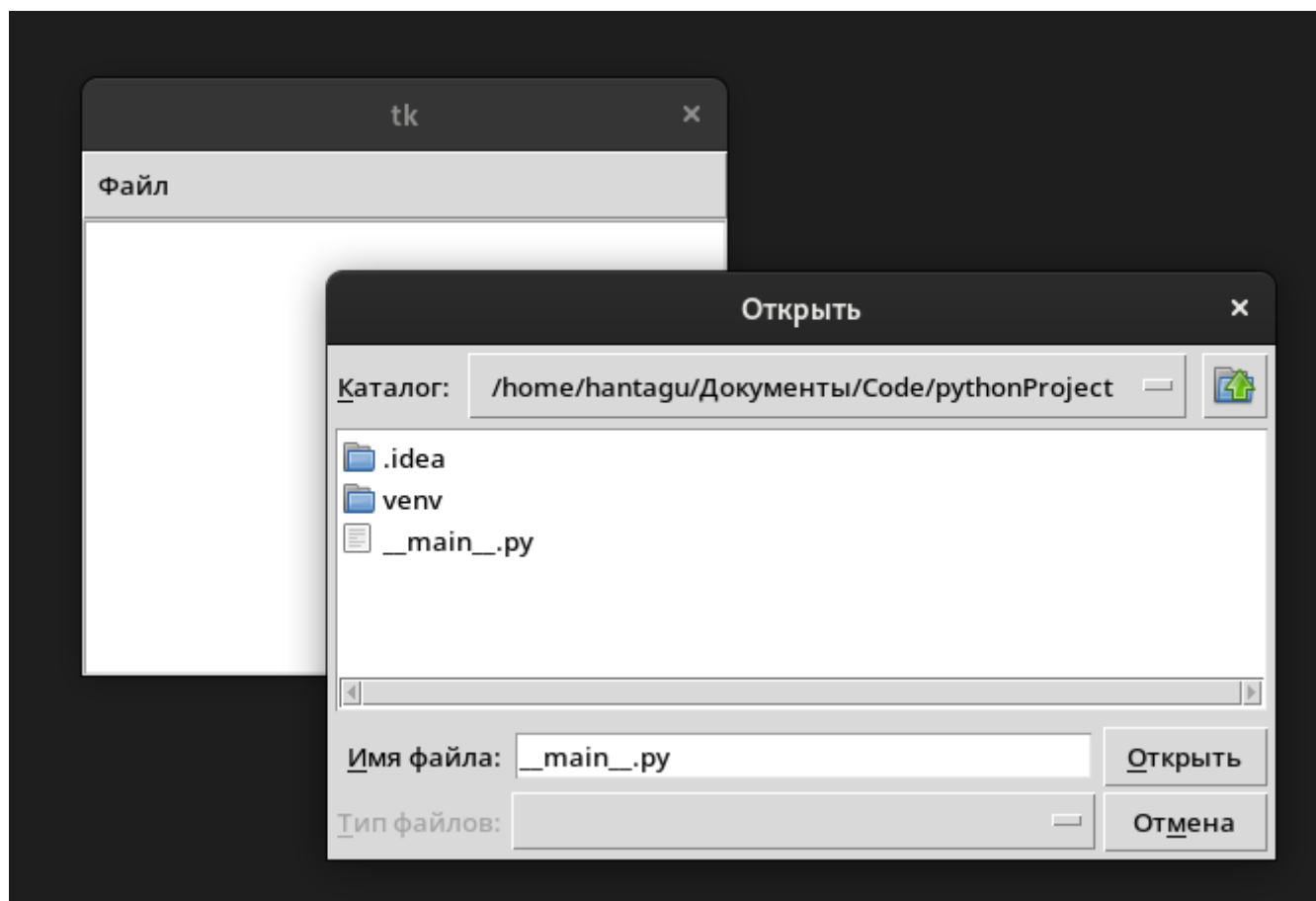


Рисунок 21. Выполненное Задание 15

Задание 16. Написать программу-викторину. Викторина должна состоять из нескольких вопросов с вариантами ответов. Вопросы должны храниться в отдельном файле. Программа должна включать в себя необходимые функции для отображения вопросов, обработки ответов пользователя и подведения итогов его результатов.

```
import tkinter as tk
import random
import csv

def show_question(question_num):
    global current_question
    current_question = question_num
    question_lbl.config(text=f"{current_question+1}/{num_of_questions}.
{questions[question_num][0]}")
    for i in range(4):
        answer_btns[i].config(text=questions[question_num][1][i])
def check_answer(answer):
    results.append(answer == questions[current_question][2])
    if current_question < num_of_questions-1:
        show_question(current_question+1)
    else:
        show_results()
def show_questions():
    questions_window = tk.Toplevel(root)
    questions_window.title("Вопросы")
    questions_text = tk.Text(questions_window, height=10, width=80, wrap='word')
    for i in range(num_of_questions):
        questions_text.insert(tk.END, f"{questions[i][0]}: {'Правильно' if results[i]
else 'Неправильно'}\n")
    questions_text.config(state='disabled')
    questions_text.pack()
def show_results():
```

```

    result_window = tk.Toplevel(root)
    result_window.title("Результаты викторины")
    tk.Label(result_window, text=f"Ваш результат:
{sum(results)}/{num_of_questions}").pack()
    tk.Button(result_window, text=f"Посмотреть вопросы",
command=show_questions).pack()
root = tk.Tk()
root.title("Викторина")
root.geometry("520x210")
with open('questions.csv') as csvfile:
    reader = csv.reader(csvfile)
    rows = list(reader)
questions = []
results = []
current_question = 0
num_of_questions = 5
for row in rows[1:]:
    questions.append([row[0], row[1:5], row[5]])
random.shuffle(questions)
question_lbl = tk.Label(root, text="question")
answer_btns = []
for i in range(4):
    answer_btns.append(tk.Button(root, text="", command=lambda i=i:
check_answer(questions[current_question][1][i])))
total_lbl = tk.Label(root, text="")
question_lbl.pack(pady=10)
for i in range(4):
    answer_btns[i].pack(pady=5)
total_lbl.pack(pady=10)
show_question(0)

```

```
root.mainloop()
```

Вопросы хранятся в структурированном файле формата CSV. Каждый вопрос представлен в виде строки, содержащей следующие заголовки: «Question» (вопрос), «Option1», «Option2», «Option3», «Option4» (варианты ответов) и «Answer» (правильный ответ на вопрос). Такая структура файла позволяет удобно и эффективно хранить и обрабатывать большое количество вопросов и ответов на них.

Пример файла с вопросами:

Question,Option1,Option2,Option3,Option4,Answer

Сколько планет в Солнечной системе?,8,9,7,10,8

Какое государство самое большое по площади?,ОАЭ,КНР,Россия,США,Россия

Сколько дней в феврале в високосный год?,29,28,30,31,29

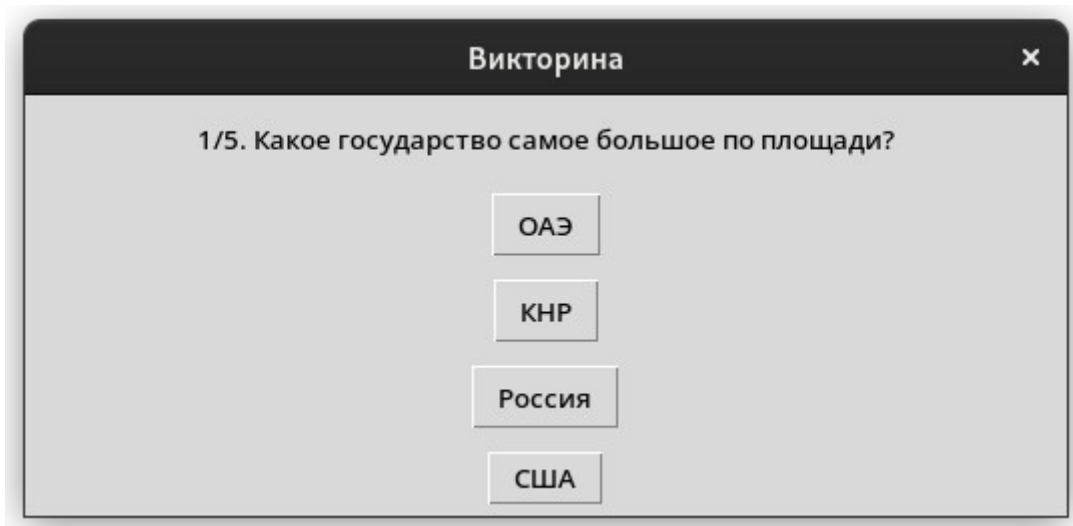


Рисунок 22. Окно викторины

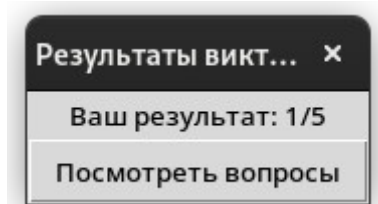


Рисунок 23. Окно с
результатами
викторины

Техника работы с библиотекой NumPy

NumPy (Numerical Python) — библиотека для языка программирования Python, которая предоставляет поддержку для работы с многомерными массивами и матрицами, а также различными операциями над ними. Основными функциями и методами библиотеки NumPy являются:

- Создание массивов и матриц:
 - `numpy.array()` — создает одномерный или многомерный массив из списка или кортежа;
 - `numpy.zeros()` — создает массив заполненный нулями указанной формы;
 - `numpy.ones()` — создает массив заполненный единицами указанной формы;
 - `numpy.full()` — создает массив заполненный указанным значением указанной формы;
 - `numpy.eye()` — создает единичную матрицу указанной формы;
- Операции над массивами:
 - Арифметические операции: `+`, `-`, `*`, `/`, `//`, `%`, `**`;
 - Унарные операции: `abs()`, `sqrt()`, `exp()`, `log()`, `sin()`, `cos()`, `tan()`, `arcsin()`, `arccos()`, `arctan()`;
 - Сравнение: `==`, `!=`, `<`, `<=`, `>`, `>=`;
 - Логические операции: `&`, `|`, `~`, `^`;
- Индексация и срезы:
 - Одномерные массивы: `array[index]`, `array[start:stop:step]`;
 - Многомерные массивы: `array[row_index, column_index]`, `array[start:stop:step, start:stop:step]`;
- Математические функции:
 - `numpy.sum()` — сумма элементов массива или матрицы;
 - `numpy.mean()` — среднее значение элементов массива или матрицы;
 - `numpy.std()` — стандартное отклонение элементов массива или матрицы;
 - `numpy.min()` — минимальное значение элементов массива или матрицы;
 - `numpy.max()` — максимальное значение элементов массива или матрицы;

- Работа с файлами:
 - `numru.save()`: сохраняет массивы в файл в бинарном формате;
 - `numru.load()`: загружает массивы из бинарного файла.

Это лишь некоторые из функций и методов библиотеки NumPy. Библиотека NumPy имеет широкий спектр возможностей и может использоваться в различных областях науки и инженерии.

Техника работы с библиотекой Matplotlib

Matplotlib — библиотека для создания графиков и визуализации данных в Python. Она предоставляет широкие возможности для создания различных типов графиков, включая линейные, точечные, столбчатые, гистограммы, 3D-графики и многое другое. Основные функции и методы, предоставляемые библиотекой Matplotlib, включают в себя:

- `plt.plot(x, y, label='label')` — создает линейный график с заданными значениями `x` и `y`, а также добавляет метку для легенды;
- `plt.scatter(x, y, s=None, c=None, marker=None, cmap=None)` — создает точечный график с заданными значениями `x` и `y`, а также задает размер точек, цвет точек и тип маркера;
- `plt.bar(x, height, width=0.8, bottom=None, align='center', data=None)` — создает столбчатую диаграмму с заданными значениями высоты столбцов и шириной столбцов;
- `plt.hist(x, bins=None, density=False, histtype='bar', color=None, label=None)` — создает гистограмму на основе заданных значений, а также задает количество бинов, тип гистограммы и цвет;
- `plt.imshow(X, cmap=None)` — отображает изображение на основе массива `X`, а также задает цветовую карту (`colormap`);
- `plt.contour(X, Y, Z, levels=None, cmap=None)` — создает контурный график на основе заданных значений `X`, `Y` и `Z`, а также задает уровни и цветовую карту;
- `plt.subplot(nrows, ncols, index)` — создает сетку из нескольких графиков в одном изображении, где `nrows` и `ncols` задают количество строк и столбцов в сетке, а `index` задает текущий индекс графика;
- `plt.legend()` — добавляет легенду на график;
- `plt.xlabel()` и `plt.ylabel()` — задают подписи для осей `x` и `y` соответственно;
- `plt.title()` — задает заголовок для графика.

Это только небольшой список функций и методов, которые предоставляет библиотека Matplotlib. Она также имеет множество других функций и настроек, которые позволяют более тонко настраивать визуализацию данных и создавать качественные и информативные графики.

Задание 1. В текстовом файле заданы координаты для более чем 10 точек на плоскости. Построить все возможные треугольники из множества данных точек. Вывести на экран координаты точек и треугольника с самой большой площадью.

```
import itertools
import matplotlib.pyplot as plt
def s(a, b, c):
    return .5 * abs((b[0]-a[0])*(c[1]-a[1]) - (c[0]-a[0])*(b[1]-a[1]))
with open('input.txt') as f:
    coords = [tuple(map(int, row.strip().split())) for row in f]
m = [None, 0]
for a, b, c in itertools.combinations(coords, 3):
    plt.plot([a[0], b[0], c[0]], [a[1], b[1], c[1]], '.')
    sq = s(a, b, c)
    if sq > m[1]:
        m[0] = [a, b, c]
        m[1] = sq
m[0].append(m[0][0])
xs, ys = zip(*m[0])
plt.plot(xs, ys)
plt.axis([-50, 50, -50, 50])
plt.savefig('output.png')
```

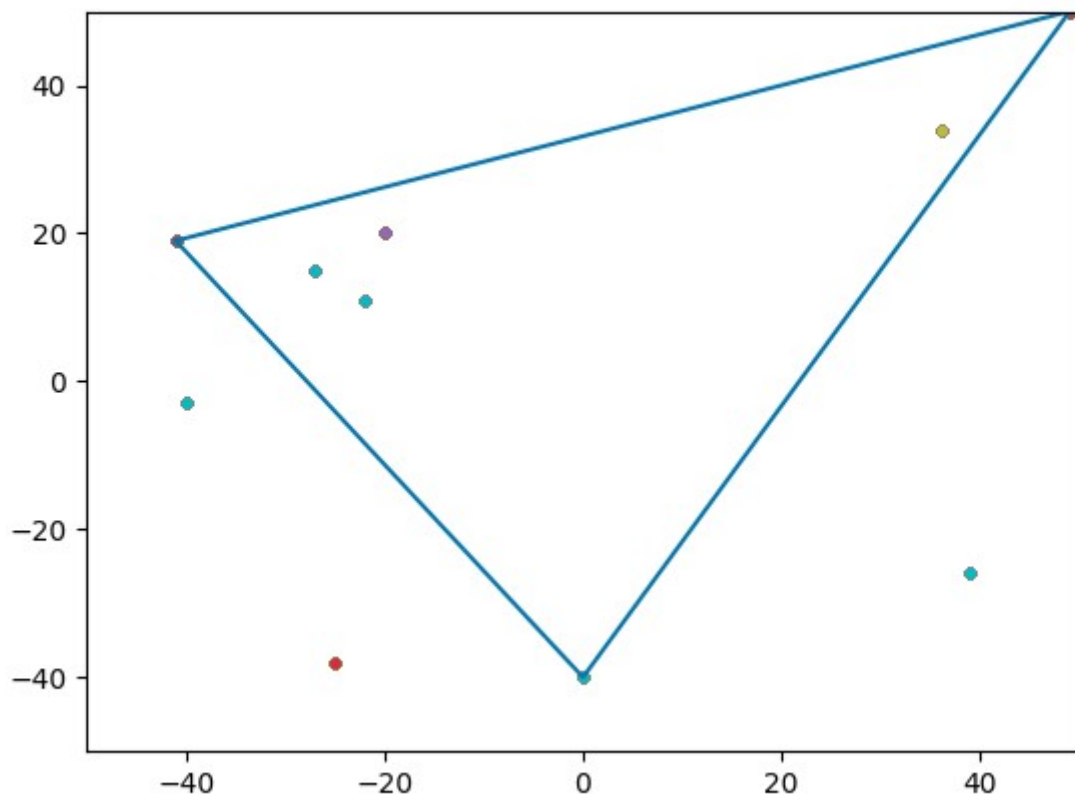


Рисунок 24. Выполненное Задание 1

Техника работы с библиотекой PyQt

PyQt — набор Python-интерфейсов для библиотеки Qt, которая представляет собой кроссплатформенный фреймворк для разработки графических интерфейсов пользователя. PyQt позволяет создавать приложения с помощью Python, используя инструменты, которые Qt предоставляет для разработки на C++. Основные функции и методы PyQt включают в себя:

- QtWidgets — модуль, который предоставляет набор виджетов, используемых для создания пользовательских интерфейсов. Некоторые из этих виджетов включают в себя кнопки, таблицы, диалоговые окна, меню, текстовые поля и т. д.;
- QtCore — модуль, который содержит базовые классы и функции для работы с Qt. Например, классы Qt для работы с событиями, потоками, таймерами, файлами и т. д.;
- QtGui — модуль, который содержит дополнительные классы и функции для работы с графическими элементами интерфейса. Например, классы для работы с рисованием, шрифтами, изображениями и т. д.;
- QtDesigner — инструмент для создания графического интерфейса пользователя с помощью графического интерфейса. Он позволяет создавать интерфейсы с помощью перетаскивания и размещения виджетов;
- Signals and Slots — система взаимодействия между виджетами, которая позволяет создавать пользовательские сигналы и связывать их с функциями-обработчиками;
- Layouts — система расположения виджетов на форме, которая позволяет автоматически управлять размером и позицией виджетов;
- Stylesheets — механизм, позволяющий изменять внешний вид виджетов, используя каскадные таблицы стилей.

Это только некоторые из основных функций и методов, которые предоставляет PyQt. Библиотека имеет богатый набор инструментов и функциональности для создания мощных графических интерфейсов пользователя в приложениях на Python.

Задание 1. Текст задания. С помощью программы Qt Designer создать интерфейс по шаблону и запустить программу. Результат выполнения задания на Рисунке 25

```
from PyQt5 import uic
from PyQt5.QtWidgets import QApplication
Form, Window = uic.loadUiType("tracker.ui")
app = QApplication([])
window = Window()
form = Form()
form.setupUi(window)
window.show()
app.exec_()
```

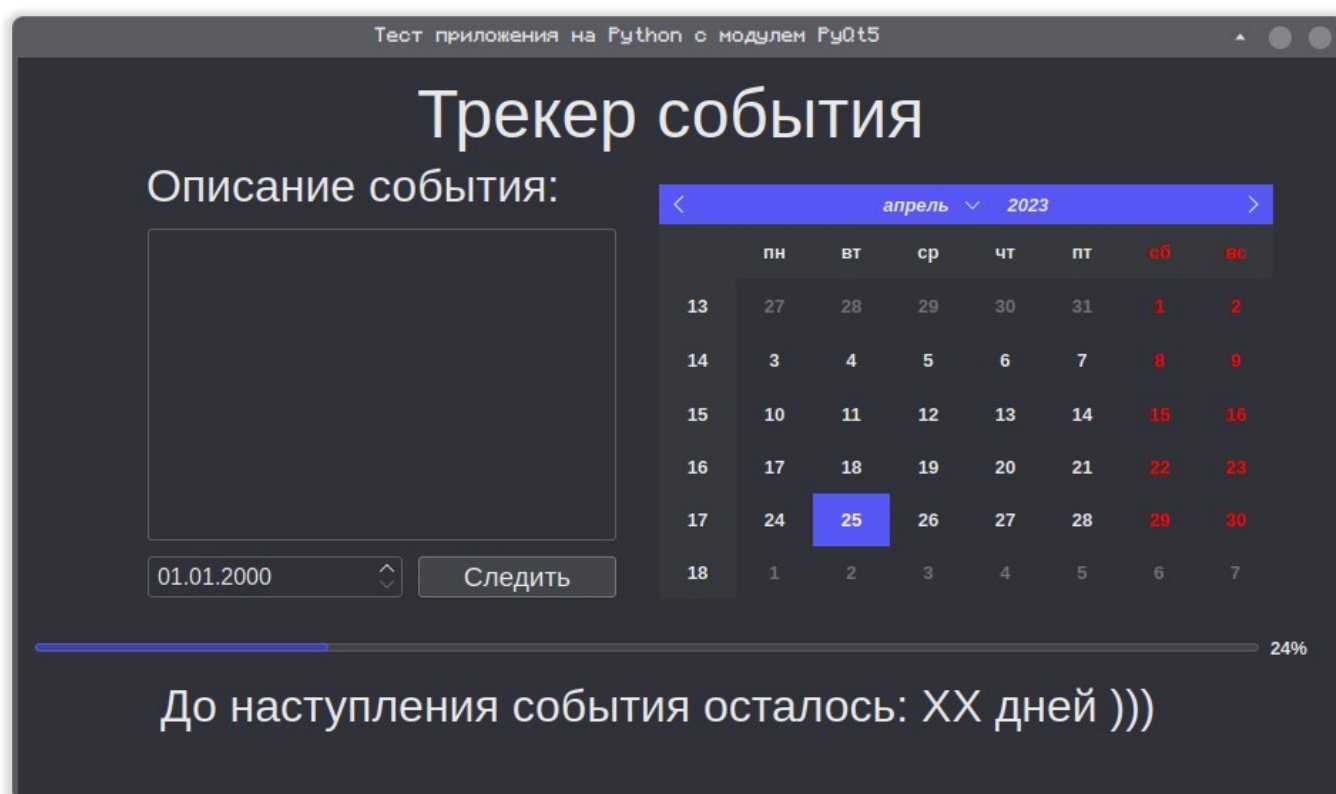


Рисунок 25. Выполненное Задание 1

Задание 2. С помощью команды `python -m PyQt5.uic.pyuic -x tracker.ui -o tracker.py` преобразовать файл созданный с помощью программы Qt Designer в программу на языке программирования Python.

```
# -*- coding: utf-8 -*-
# Form implementation generated from reading ui file 'tracker.ui'
# Created by: PyQt5 UI code generator 5.15.9
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again. Do not edit this file unless you know what you are doing.
from PyQt5 import QtCore, QtGui, QtWidgets
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(928, 518)
        font = QtGui.QFont()
        font.setFamily("Liberation Sans")
        font.setPointSize(10)
        font.setBold(True)
        font.setItalic(True)
        font.setWeight(75)
        MainWindow.setFont(font)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        font = QtGui.QFont()
        font.setFamily("Liberation Sans")
        font.setPointSize(10)
        font.setBold(True)
        font.setItalic(True)
        font.setWeight(75)
        self.centralwidget.setFont(font)
        self.centralwidget.setObjectName("centralwidget")
        self.label = QtWidgets.QLabel(self.centralwidget)
```

```
self.label.setGeometry(QtCore.QRect(280, 10, 361, 61))
font = QtGui.QFont()
font.setFamily("Liberation Sans")
font.setPointSize(36)
font.setBold(False)
font.setItalic(False)
font.setWeight(50)
self.label.setFont(font)
self.label.setObjectName("label")
self.calendarWidget = QtWidgets.QCalendarWidget(self.centralwidget)
self.calendarWidget.setGeometry(QtCore.QRect(450, 90, 431, 291))
font = QtGui.QFont()
font.setFamily("Liberation Sans")
font.setPointSize(10)
font.setBold(True)
font.setItalic(False)
font.setWeight(75)
self.calendarWidget.setFont(font)
self.calendarWidget.setObjectName("calendarWidget")
self.progressBar = QtWidgets.QProgressBar(self.centralwidget)
self.progressBar.setGeometry(QtCore.QRect(10, 390, 901, 51))
font = QtGui.QFont()
font.setFamily("Liberation Sans")
font.setPointSize(10)
font.setBold(True)
font.setItalic(False)
font.setWeight(75)
self.progressBar.setFont(font)
self.progressBar.setProperty("value", 24)
self.progressBar.setObjectName("progressBar")
```

```

self.plainTextEdit = QtWidgets.QPlainTextEdit(self.centralwidget)
self.plainTextEdit.setGeometry(QtCore.QRect(90, 120, 331, 221))
self.plainTextEdit.setObjectName("plainTextEdit")
self.label_2 = QtWidgets.QLabel(self.centralwidget)
self.label_2.setGeometry(QtCore.QRect(90, 70, 291, 41))
font = QtGui.QFont()
font.setFamily("Liberation Sans")
font.setPointSize(24)
font.setBold(False)
font.setItalic(False)
font.setWeight(50)
self.label_2.setFont(font)
self.label_2.setObjectName("label_2")
self.pushButton = QtWidgets.QPushButton(self.centralwidget)
self.pushButton.setGeometry(QtCore.QRect(280, 350, 141, 31))
font = QtGui.QFont()
font.setFamily("Liberation Sans")
font.setPointSize(14)
font.setBold(False)
font.setItalic(False)
font.setWeight(50)
self.pushButton.setFont(font)
self.pushButton.setObjectName("pushButton")
self.dateEdit = QtWidgets.QDateEdit(self.centralwidget)
self.dateEdit.setGeometry(QtCore.QRect(90, 350, 181, 31))
font = QtGui.QFont()
font.setFamily("Liberation Sans")
font.setPointSize(12)
font.setBold(False)
font.setItalic(False)

```

```

font.setWeight(50)
self.dateEdit.setFont(font)
self.dateEdit.setObjectName("dateEdit")
self.label_3 = QtWidgets.QLabel(self.centralwidget)
self.label_3.setGeometry(QtCore.QRect(100, 430, 721, 51))
font = QtGui.QFont()
font.setFamily("Liberation Sans")
font.setPointSize(24)
font.setBold(False)
font.setItalic(False)
font.setWeight(50)
self.label_3.setFont(font)
self.label_3.setObjectName("label_3")
MainWindow.setCentralWidget(self.centralwidget)
self.statusbar = QtWidgets.QStatusBar(MainWindow)
self.statusbar.setObjectName("statusbar")
MainWindow.setStatusBar(self.statusbar)
self.retranslateUi(MainWindow)
QtCore.QMetaObject.connectSlotsByName(MainWindow)
def retranslateUi(self, MainWindow):
    _translate = QtCore.QCoreApplication.translate
    MainWindow.setWindowTitle(_translate("MainWindow", "Тест
приложения на Python с модулем PyQt5"))
    self.label.setText(_translate("MainWindow", "Трекер события"))
    self.label_2.setText(_translate("MainWindow", "Описание события:"))
    self.pushButton.setText(_translate("MainWindow", "Следить"))
    self.label_3.setText(_translate("MainWindow", "До наступления события
осталось: XX дней )))"))
if __name__ == "__main__":
    import sys

```

```
app = QtWidgets.QApplication(sys.argv)
MainWindow = QtWidgets.QMainWindow()
ui = Ui_MainWindow()
ui.setupUi(MainWindow)
MainWindow.show()
sys.exit(app.exec_())
```

Техника работы с библиотекой PyGame

Pygame — кроссплатформенная библиотека на языке программирования Python, которая предоставляет инструменты для создания игр и мультимедийных приложений. Она содержит множество функций и методов для работы с изображениями, звуком, анимацией, клавиатурой, мышью и т. д. Основными функциями и методами библиотеки Pygame являются:

- `pygame.init()` — инициализация Pygame, вызывается в начале программы;
- `pygame.display.set_mode()` — создание окна приложения;
- `pygame.display.set_caption()` — задание названия окна;
- `pygame.image.load()` — загрузка изображения;
- `pygame.Surface.blit()` — отображение изображения на экране;
- `pygame.time.Clock()` — создание объекта для работы со временем;
- `pygame.event.get()` — получение списка событий;
- `pygame.event.poll()` — получение одного события;
- `pygame.key.get_pressed()` — получение состояния клавиш на клавиатуре;
- `pygame.mouse.get_pos()` — получение координат курсора мыши;
- `pygame.mixer.Sound()` — загрузка звукового файла;
- `pygame.mixer.Sound.play()` — воспроизведение звука.

Кроме того, Pygame содержит множество других функций и методов, таких как работа с спрайтами, рисование на экране, создание анимации, обнаружение столкновений и т. д. Благодаря этим функциям и методам, Pygame позволяет создавать разнообразные игры и мультимедийные приложения на языке Python.

Задание 1. Написать программу, которая представляет собой инвентарь с несколькими слотами. Пользователь должен иметь возможность перемещать предметы в инвентаре с помощью мыши. Файл с текстурой должен быть размером 60 на 60 пикселей и иметь имя bread.png.

```
import pygame

SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600
SCREEN_SIZE = (SCREEN_WIDTH, SCREEN_HEIGHT)
CELL_SIZE = 64
CELL_PADDING = 5
CELL_BORDER_WIDTH = 2
CELL_BORDER_COLOR = (255, 255, 255)
SELECTED_CELL_BORDER_COLOR = (255, 255, 0)
BACKGROUND_COLOR = (0, 0, 0)
TEXT_COLOR = (255, 255, 255)
FONT_SIZE = 20
FONT_NAME = pygame.font.match_font('arial')
ROWS = 3
COLUMNS = 9

def draw_text(surface, text, x, y, color):
    font = pygame.font.Font(FONT_NAME, FONT_SIZE)
    text_surface = font.render(text, True, color)
    text_rect = text_surface.get_rect()
    text_rect.midtop = (x, y)
    surface.blit(text_surface, text_rect)

def get_cell_index(x, y):
    row = (y - CELL_PADDING) // (CELL_SIZE + CELL_PADDING)
    column = (x - CELL_PADDING) // (CELL_SIZE + CELL_PADDING)
    return row * COLUMNS + column

pygame.init()
```

```

screen = pygame.display.set_mode(SCREEN_SIZE)
pygame.display.set_caption('Minecraft Inventory')
inventory = []
for row in range(ROWS):
    for column in range(COLUMNS):
        x = (CELL_SIZE + CELL_PADDING) * column + CELL_PADDING
        y = (CELL_SIZE + CELL_PADDING) * row + CELL_PADDING
        inventory.append(pygame.Rect(x, y, CELL_SIZE, CELL_SIZE))
items = [None] * len(inventory)
items[0] = pygame.image.load('bread.png')
selected_cell = None
selected_item = None
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.MOUSEBUTTONDOWN:
            cell_index = get_cell_index(*event.pos)
            if cell_index >= 0 and cell_index < len(inventory):
                if event.button == 1:
                    if items[cell_index] is None:
                        if selected_item is not None:
                            items[cell_index] = selected_item
                            selected_item = None
                    else:
                        if selected_item is None:
                            selected_item = items[cell_index]
                            items[cell_index] = None
                        else:

```



```

        items[cell_index], selected_item = selected_item,
items[cell_index]
    elif event.button == 3:
        if items[cell_index] is not None:
            if selected_item is None:
                selected_item = items[cell_index]
                items[cell_index] = None
    screen.fill(BACKGROUND_COLOR)
    for i, rect in enumerate(inventory):
        pygame.draw.rect(screen, CELL_BORDER_COLOR, rect,
CELL_BORDER_WIDTH)
        if items[i] is not None:
            screen.blit(items[i], rect)
        if rect == selected_cell:
            pygame.draw.rect(screen, SELECTED_CELL_BORDER_COLOR, rect,
CELL_BORDER_WIDTH)
            draw_text(screen, 'Selected item: {}'.format(selected_item), SCREEN_WIDTH //
2, CELL_PADDING, TEXT_COLOR)
    pygame.display.flip()
pygame.quit()

```

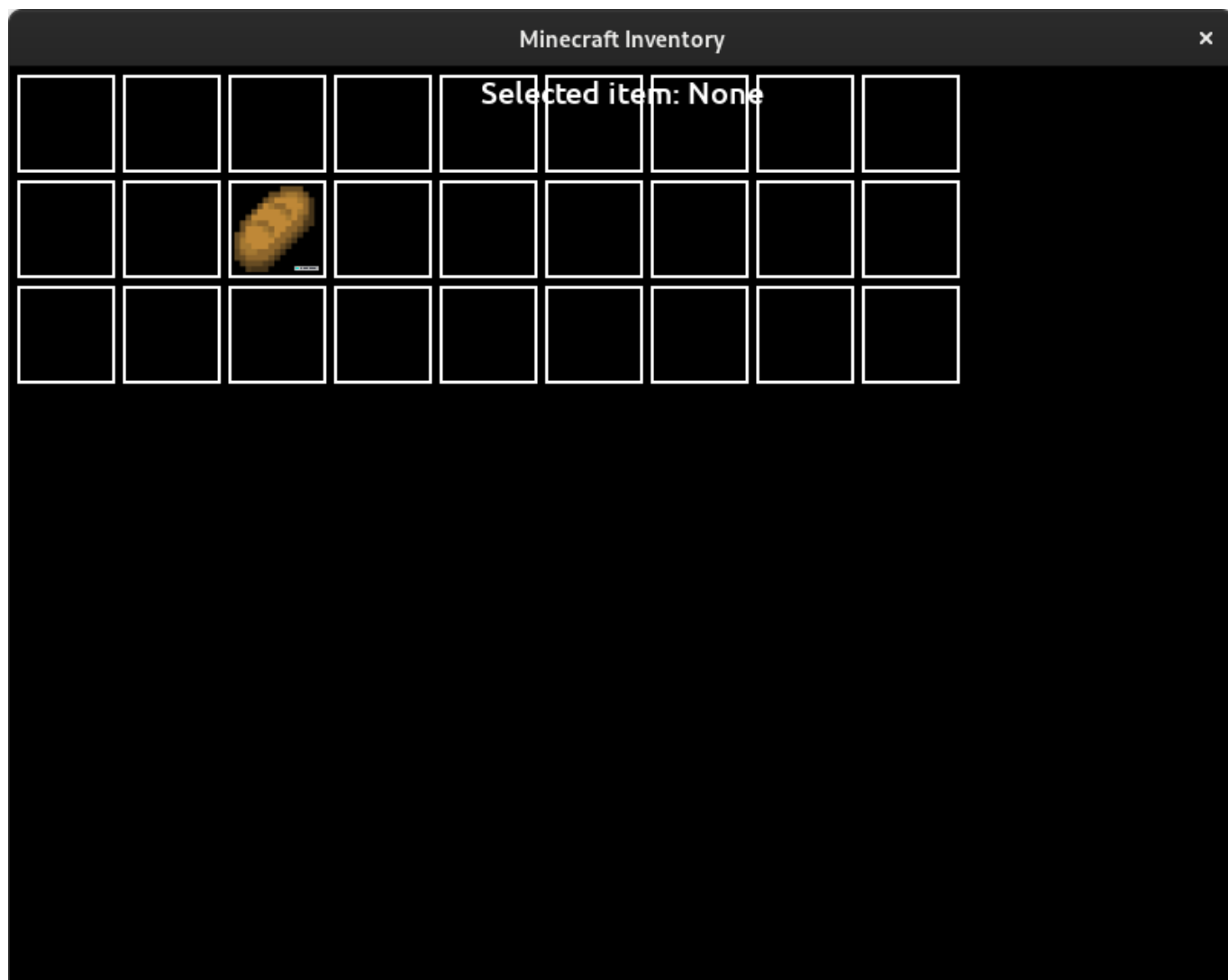


Рисунок 26. Выполненное Задание 1

Техника работы с базами данных

SQLite — встраиваемая реляционная база данных, которая может быть использована в приложениях Python с помощью стандартной библиотеки `sqlite3`. Библиотека `sqlite3` предоставляет набор функций для работы с базой данных SQLite из Python. Вот несколько примеров того, что можно сделать с этой библиотекой:

- Установка соединения с базой данных;
- Создание таблицы;
- Вставка данных;
- Выборка данных;
- Закрытие соединения.

SQLite предоставляет простой и легковесный способ хранения данных в приложении Python без необходимости настройки отдельного сервера базы данных. Однако, стоит заметить, что SQLite не подходит для всех сценариев использования, в частности, для больших приложений с высокой нагрузкой, для которых лучше использовать более мощные СУБД, такие как PostgreSQL или MySQL.

Задание 1. Используя библиотеки Tkinter и SQLite3, создать программу-викторину, которая будет задавать вопросы из базы данных

```
import sqlite3
import random
import tkinter as tk

class Quiz:
    def __init__(self, questions):
        self.questions = questions
        self.current_question = None
        self.score = 0

    def get_next_question(self):
        self.current_question = random.choice(self.questions)
        return self.current_question

    def check_answer(self, selected):
        if selected == int(self.current_question["answer"][0]):
            self.score += 1

    def get_score(self):
        return self.score

class QuizApp:
    def __init__(self, master):
        self.master = master
        self.master.title("Викторина")
        self.quiz = None
        self.question_label = tk.Label(self.master, text="", font=("Helvetica", 16))
        self.question_label.pack(pady=10)
        self.answer_button1 = tk.Button(self.master, text="", font=("Helvetica", 12),
command=lambda: self.check_answer(1))
        self.answer_button1.pack(pady=5)
        self.answer_button2 = tk.Button(self.master, text="", font=("Helvetica", 12),
command=lambda: self.check_answer(2))
```

```

        self.answer_button2.pack(pady=5)
        self.answer_button3 = tk.Button(self.master, text="", font=("Helvetica", 12),
command=lambda: self.check_answer(3))
        self.answer_button3.pack(pady=5)
        self.answer_button4 = tk.Button(self.master, text="", font=("Helvetica", 12),
command=lambda: self.check_answer(4))
        self.answer_button4.pack(pady=5)
        self.score_label = tk.Label(self.master, text="", font=("Helvetica", 16))
        self.score_label.pack(pady=10)
        self.start_button = tk.Button(self.master, text="Начать викторину",
font=("Helvetica", 14), command=self.start_quiz)
        self.start_button.pack(pady=10)
        self.conn = sqlite3.connect(':memory:')
        self.put_questions()
    def start_quiz(self):
        self.quiz = Quiz(self.get_questions())
        self.score_label.config(text="")
        self.start_button.config(text="Следующий вопрос",
command=self.next_question)
        self.next_question()
    def next_question(self):
        if self.quiz.get_score() == 10:
            self.end_quiz()
        else:
            self.question = self.quiz.get_next_question()
            self.question_label.config(text=self.question["question"])
            self.answer_button1.config(text=self.question["options"][0])
            self.answer_button2.config(text=self.question["options"][1])
            self.answer_button3.config(text=self.question["options"][2])
            self.answer_button4.config(text=self.question["options"][3])

```

```

def check_answer(self, selected):
    self.quiz.check_answer(selected)
    self.score_label.config(text="Счет: {}".format(self.quiz.get_score()))
    self.next_question()
def end_quiz(self):
    self.question_label.config(text="Конец викторины")
    self.answer_button1.config(text="")
    self.answer_button2.config(text="")
    self.answer_button3.config(text="")
    self.answer_button4.config(text="")
    self.score_label.config(text="Ваш итоговый счет:
{}".format(self.quiz.get_score()))
    self.start_button.config(text="Начать викторину", command=self.start_quiz)
def get_questions(self):
    cursor = self.conn.cursor()
    cursor.execute("SELECT * FROM questions ORDER BY RANDOM() LIMIT
10")

    rows = cursor.fetchall()
    questions = []
    for row in rows:
        question = {
            "question": row[0],
            "options": [row[1], row[2], row[3], row[4]],
            "answer": row[5]
        }
        questions.append(question)
    return questions
def put_questions(self):
    cursor = self.conn.cursor()

```

```
cursor.execute("CREATE TABLE IF NOT EXISTS questions (question TEXT,  
option1 TEXT, option2 TEXT, option3 TEXT, option4 TEXT, answer TEXT);")
```

```
cursor.execute("INSERT INTO questions (question, option1, option2, option3,  
option4, answer) VALUES ('Какое имя было у дракона в мультфильме \"Мулан\"?', '1.  
Мушу', '2. Чи-Фу', '3. Шань-Ю', '4. Крикет', '1. Мушу');")
```

```
cursor.execute("INSERT INTO questions (question, option1, option2, option3,  
option4, answer) VALUES ('Кто создал первую версию операционной системы  
Windows?', '1. Билл Гейтс', '2. Стив Джобс', '3. Тим Бернерс-Ли', '4. Алан Тьюринг', '1.  
Билл Гейтс');")
```

```
cursor.execute("INSERT INTO questions (question, option1, option2, option3,  
option4, answer) VALUES ('Кто написал произведение \"Война и мир\"?', '1. Лев  
Толстой', '2. Федор Достоевский', '3. Антон Чехов', '4. Николай Гоголь', '1. Лев  
Толстой');")
```

```
cursor.execute("INSERT INTO questions (question, option1, option2, option3,  
option4, answer) VALUES ('Кто сыграл главную роль в фильме \"Титаник\"?', '1.  
Леонардо ДиКаприо', '2. Джонни Депп', '3. Брэд Питт', '4. Том Круз', '1. Леонардо  
ДиКаприо');")
```

```
cursor.execute("INSERT INTO questions (question, option1, option2, option3,  
option4, answer) VALUES ('Какой город является столицей России?', '1. Москва', '2.  
Санкт-Петербург', '3. Екатеринбург', '4. Новосибирск', '1. Москва');")
```

```
cursor.execute("INSERT INTO questions (question, option1, option2, option3,  
option4, answer) VALUES ('Кто написал произведение \"Преступление и  
наказание\"?', '1. Федор Достоевский', '2. Лев Толстой', '3. Антон Чехов', '4. Николай  
Гоголь', '1. Федор Достоевский');")
```

```
cursor.execute("INSERT INTO questions (question, option1, option2, option3,  
option4, answer) VALUES ('Кто является автором произведения \"Ромео и  
Джульетта\"?', '1. Уильям Шекспир', '2. Федор Достоевский', '3. Лев Толстой', '4.  
Александр Пушкин', '1. Уильям Шекспир');")
```

```
cursor.execute("INSERT INTO questions (question, option1, option2, option3,  
option4, answer) VALUES ('Кто является главным персонажем
```

мультфильма \"Шрек\"?', '1. Шрек', '2. Фиона', '3. Осел', '4. Кот в сапогах', '1.
Шрек');")

```
if __name__ == "__main__":
```

```
    root = tk.Tk()
```

```
    app = QuizApp(root)
```

```
    root.mainloop()
```

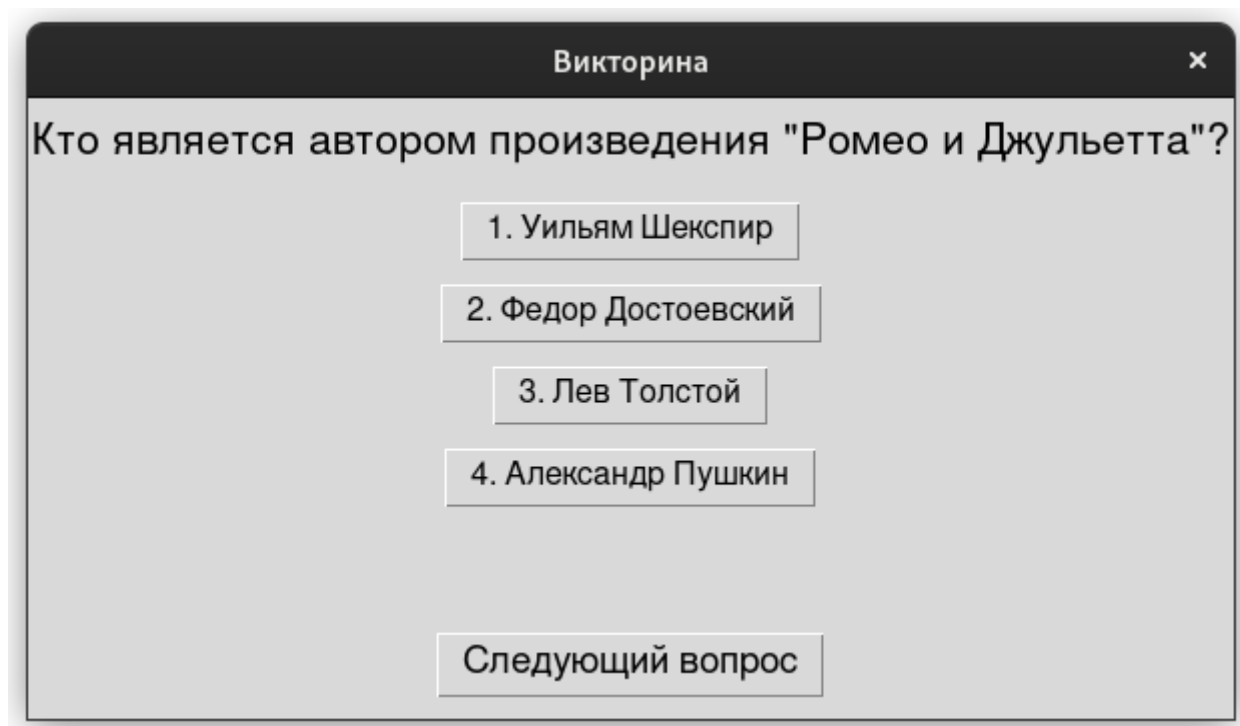



Рисунок 27. Выполненное Задание 1