МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение  
 высшего образования

«АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Колледж Алтайского государственного университета

Отделение экономики и информационных технологий

ОТЧЕТ

по дисциплине «Прикладное программирование»

СОЗДАНИЕ ПРИЛОЖЕНИЯ С БАЗОЙ ДАННЫХ И ГРАФИЧЕСКИМ ИНТРФЕЙСОМ

Выполнил студент

2 курса 283б-сп группы

Бибиков И.В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Научный руководитель

Преподаватель

Овсянников А.В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Работа защищена

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2020г.

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Барнаул 2020

СОДЕРЖАНИЕ

[1. СОЗДАНИЕ ПРИЛОЖЕНИЯ 3](#_Toc40357766)

[1.1 Создание главного окна 3](#_Toc40357767)

[1.2. Создание дочерних окон 5](#_Toc40357768)

[1.3.Добавление виджетов 7](#_Toc40357769)

[1.4.Создание базы данных и добавление информации в нее. 9](#_Toc40357770)

[1.5.Редактирование данных в таблице базы SQLite. 12](#_Toc40357771)

[1.6.Удаление данных в таблице базы SQLite 14](#_Toc40357772)

[2.РАБОТА ПРОГРАММЫ 16](#_Toc40357773)

[2.1. Добавление данных 16](#_Toc40357774)

[2.2.Редактирование данных 17](#_Toc40357775)

[2.3.Удаление данных 18](#_Toc40357776)

[ПРИЛОЖЕНИЕ 1 20](#_Toc40357777)

ВВЕДЕНИЕ

Создать приложение с базой данных и графическим интерфейсом Tkinter на Python.Создать функции для работоспособности базы с приложением.

Сделать необходимые запросы и соединить с функциями.

# 1. СОЗДАНИЕ ПРИЛОЖЕНИЯ

# 1.1 Создание главного окна

Для начала мы реализуем главное окно программы:

import tkinter as tk

Создаем класс главного окна и конструктор:

class Main(tk.Frame):

def \_\_init\_\_(self, root):

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

Метод super() – отыскивает базовый класс класса Main() и возвращает его, а дальше идет обращение к методу \_init\_ этого найденного класса. Польза в том что при изменении родительского класса не придётся изменять содержимое метода ,а также дает возможность корректно использовать класс в наследовании.

И делаем сводную конструкцию:

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

root = tk.Tk()

app = Main(root)

app.pack()

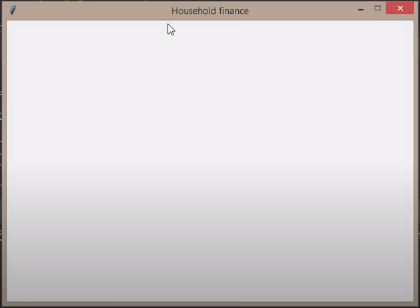
root.title("Домашние финансы")

root.geometry("650x450+300+200")

root.resizable(False, False)

root.mainloop()

В 1 раз запускаем программу и видим пустое созданное окно:



# 1.2. Создание дочерних окон

Для начала создадим функцию def init\_main(self): в которой мы будем хранить и инициализировать все элементы графического интерфейса. Вызывать данную функцию будем через self.init\_main().

Получается такой код:

import tkinter as tk

class Main(tk.Frame):

def \_\_init\_\_(self, root):

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

self.init\_main()

def init\_main(self):

Далее создаем инструментальную панель, которая будет занимать верхнюю часть окна и на нем будут находится кнопки, которыми мы и будем взаимодействовать с программой:

Вводим следующий код в функцию:

def init\_main(self):

toolbar = tk.Frame(bg='#d7d8e0', bd=2) \ цвет панели и граница

toolbar.pack(side=tk.TOP, fill=tk.X) \для закрепления панели

Создадим новый класс создания дочернего окна и определим конструктор:

class Child(tk.Toplevel):

def \_\_init\_\_(self):

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

Далее напишем функцию, в которой мы будем инициализировать функции окна.

def init\_child(self):

Укажем имя, размер дочернего окна, и чтобы его размеры были неизменны, и пользователь не мог закрыть главное окно пока открыто дочернее:

self.title('Добавить доходы/расходы')

self.geometry('400x220+400+300')

self.resizable(False, False)

self.grab\_set()

self.focus\_set()

В Class main() напишем функцию которая будет отвечать за вызов дочернего окна:

def open\_dialog(self):

Child()

Также запишем в Class Child (tk.Toplevel) вызов функции:

self.init\_child()

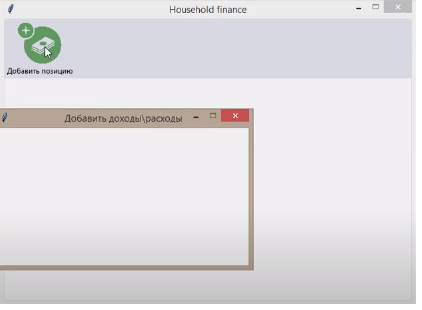
Создаем кнопку, по которой будем вызывать это окно:

self.add\_img = tk.PhotoImage(file="add.png")

btn\_open\_dialog = tk.Button(toolbar, text='Добавить позицию', command=self.open\_dialog,bg='#d7d8e0',bd=0,compound=tk.TOP,image=self.add\_img)

btn\_open\_dialog.pack(side=tk.LEFT)

Запустим программу и увидим, что есть закрепленная панель на которой находится кнопка с надписью что бы запустить дочернее окно:



И при запуске дочернего окна взаимодействовать с главным невозможно пока небудет закрыто дочернее окно.

За это отвечает метод self.grab\_set() и self.focus\_set.

* Self.grab\_set() перехватывает все события происходящие в приложении.
* Self.focus\_set() захватывает и удерживает фокус.

# Добавление виджетов

Добавим движок для создания виджетов

from tkinter import ttk

Создадим кортедж с 4 колонками, которые будут добавлены на главное окно:

self.tree = ttk.Treeview(self, columns=('ID', 'description', 'costs', 'total'), height=15, show='headings')

Обращаться к ним будем следующим образом:

self.tree.column("ID", width=30, anchor=tk.CENTER)

self.tree.column("description", width=365, anchor=tk.CENTER)

self.tree.column("costs", width=150, anchor=tk.CENTER)

self.tree.column("total", width=100, anchor=tk.CENTER)

self.tree.heading("ID", text='ID')

self.tree.heading("description", text='Наименование')

self.tree.heading("costs", text='Статья дохода/расхода')

self.tree.heading("total", text='Сумма')

Для отображения этих виджетов применяем метод:

self.tree.pack()

Далее создадим форму ввода на дочернем окне следующим кодом:

label\_description = tk.Label(self, text='Наименование:')

label\_description.place(x=50, y=50)

label\_select = tk.Label(self, text='Статья дохода/расхода:')

label\_select.place(x=50, y=80)

label\_sum = tk.Label(self, text='Сумма:')

label\_sum.place(x=50, y=110)

self.entry\_description = ttk.Entry(self)

self.entry\_description.place(x=200, y=50)

self.entry\_money = ttk.Entry(self)

self.entry\_money.place(x=200, y=110)

self.combobox = ttk.Combobox(self, values=[u"Доход", u"Расход"])

self.combobox.current(0)

self.combobox.place(x=200, y=80)

Создаем кнопки первая будет добавлять введенную информацию в полях ввода и сохранять в базы данных, а вторая будет закрывать дочернее окно.

btn\_cancel = ttk.Button(self, text='Закрыть', command=self.destroy)

btn\_cancel.place(x=300, y=170)

self.btn\_ok = ttk.Button(self, text='Добавить')

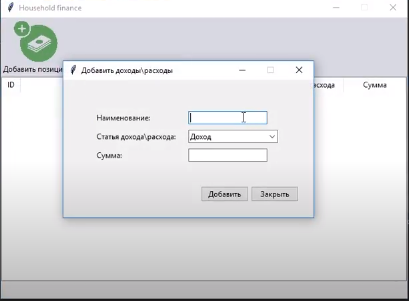
self.btn\_ok.place(x=220, y=170)

self.btn\_ok.bind('<Button-1>', lambda event: self.view.records(self.entry\_description.get(),

self.combobox.get(),

self.entry\_money.get()))

Запускаем программу и получаем следующий вид:



# Создание базы данных и добавление информации в нее.

Для создания базы данных мы импортируем модуль sql:

import sqlite3

Реализуем новый класс для работы с базой данных и конструктор класса, также делаем соединение с базой. Код получаем мы следующий:

class DB:

def \_\_init\_\_(self):

self.conn = sqlite3.connect('finance.db')

self.c = self.conn.cursor()

self.c.execute('''CREATE TABLE IF NOT EXISTS finance (id integer primary key, description text, costs text, total real)''')

self.conn.commit()

В ранее указанную функцию базы данных мы записываем:

def insert\_data(self, description, costs, total):

self.c.execute('''INSERT INTO finance(description, costs, total) VALUES (?, ?, ?)''',

(description, costs, total))

self.conn.commit()

В Class main добавляем функцию которая будет принимать значения 3 переменных:

def records(self, description, costs, total):

self.db.insert\_data(description, costs, total)

self.view\_records()

В уже созданный конструктор класса\_ main\_ добавляем. Сделанно это для того что бы мы могли обратиться к функции класса DB:

db = DB()

Помещается в данный код:

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

root = tk.Tk()

db = DB()

app = Main(root)

app.pack()

root.title("Домашние финансы")

root.geometry("650x450+300+200")

root.resizable(False, False)

root.mainloop()

Также передаем в функцию конструктора класса Main():

self.db = db

И теперь мы можем обращаться к функциям и методам класса BD из Class main через простую переменную db.

Теперь мы передаем функцию records в кнопку “добавить”:

self.btn\_ok.bind('<Button-1>', lambda event: self.view.records(self.entry\_description.get(),

self.combobox.get(),

self.entry\_money.get()))

Для того чтобы отобразить информацию с данного окна в главном окне реализуем следующую функцию:

def view\_records(self):

self.db.c.execute('''SELECT \* FROM finance''')

[self.tree.delete(i) for i in self.tree.get\_children()]

[self.tree.insert('', 'end', values=row) for row in self.db.c.fetchall()]

Для того чтобы данные автоматически передавались в базу данных мы вызовем функцию сначала в def records():

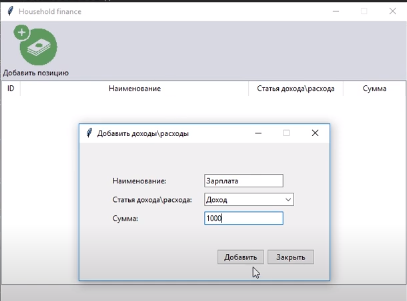
self.view\_records()

А также при запуске программы первый раз в классе Main():

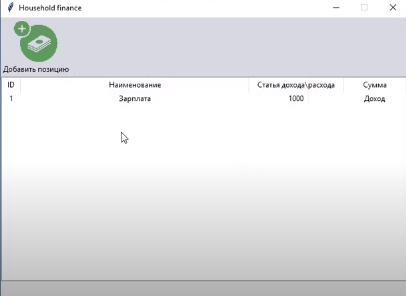
self.view\_records()

Запускаем программу проверим что получилось:

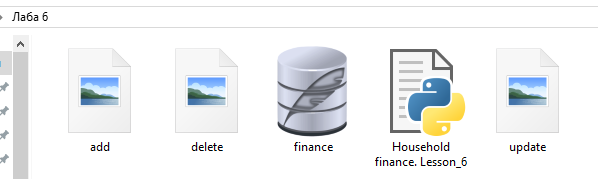
– прописываем данные в форму заполнения в дочернем окне



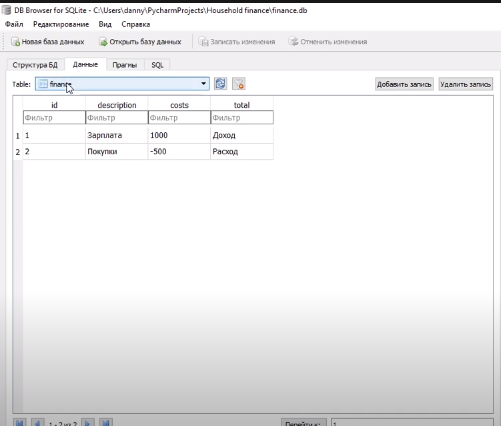
– нажимаем кнопку добавить и как можно увидеть новая запись добавилась



И теперь посмотрим создается ли сама база данных:



База данных создается и в ней действительно отображаются все данные:



# Редактирование данных в таблице базы SQLite.

В классе Main() создадим функцию:

def update\_record(self, description, costs, total):

self.db.c.execute('''UPDATE finance SET description=?, costs=?, total=? WHERE ID=?''',

(description, costs, total, self.tree.set(self.tree.selection()[0], '#1')))

self.db.conn.commit()

self.view\_records()

Создадим новый класс Update в классе Chiuld который будет иметь свой конструктор

class Update(Child):

def \_\_init\_\_(self):

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

self.init\_edit()

self.view = app

def init\_edit(self):

self.title('Редактировать позицию')

btn\_edit = ttk.Button(self, text='Редактировать')

btn\_edit.place(x=205, y=170)

btn\_edit.bind('<Button-1>', lambda event: self.view.update\_record(self.entry\_

description.get(),

self.combobox.get(),

self.entry\_money.get()))

Для того что бы изменения в графическом интерфейсе отобразились пользователю, вызовем функцию из конструктора Class Update() :

self.init\_edit()

А также чтобы обращаться к функция из Class Main() передадим в тотже конструктор класса Update():

self.view = app

Создадим функцию, которая будет выполнятьcя при нажатии кнопки «редактировать»:

def open\_update\_dialog(self):

Update()

Теперь приступаем к создаю кнопки редактировать:

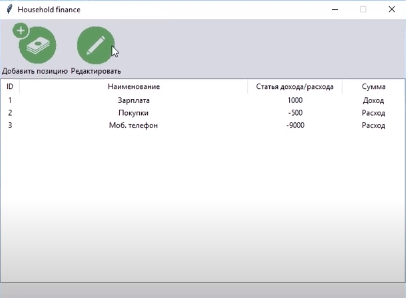
self.update\_img = tk.PhotoImage(file='update.png')

btn\_edit\_dialog = tk.Button(toolbar, text='Редактировать', bg='#d7d8e0', bd=0, image=self.update\_img,

compound=tk.TOP, command=self.open\_update\_dialog)

btn\_edit\_dialog.pack(side=tk.

Запускаем программу и видим, что появилась кнопка «Редактировать»



# Удаление данных в таблице базы SQLite

Создадим новую функцию в Class Main():

def delete\_records(self):

for selection\_item in self.tree.selection():

self.db.c.execute('''DELETE FROM finance WHERE id=?''', (self.tree.set(selection\_item, '#1')))

self.db.conn.commit()

self.view\_records()

Теперь делаем кнопку:

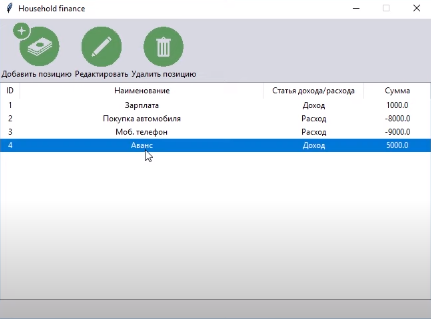
elf.delete\_img = tk.PhotoImage(file='delete.png')

btn\_delete = tk.Button(toolbar, text='Удалить позицию', bg='#d7d8e0', bd=0, image=self.delete\_img,

compound=tk.TOP, command=self.delete\_records)

btn\_delete.pack(side=tk.LEFT)

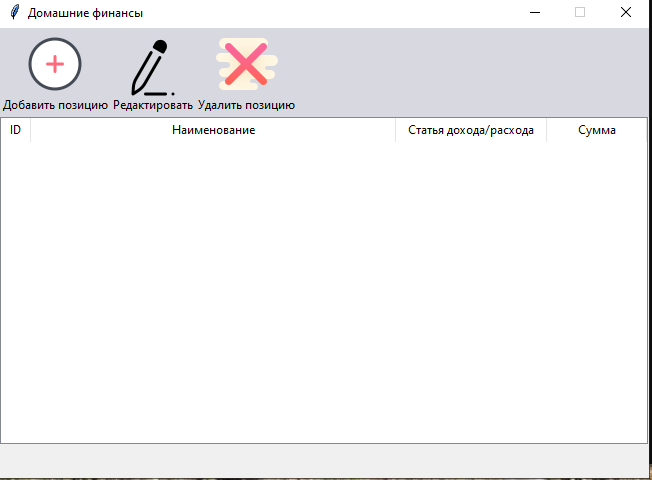
Вот сама программа:



# РАБОТА ПРОГРАММЫ

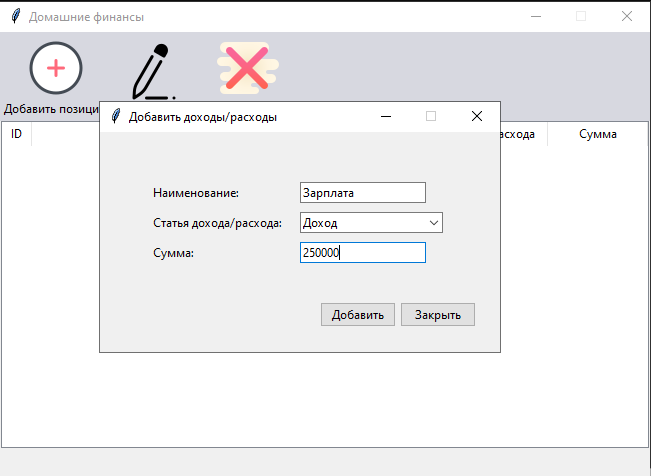
## 2.1. Добавление данных

– Запускаем программу

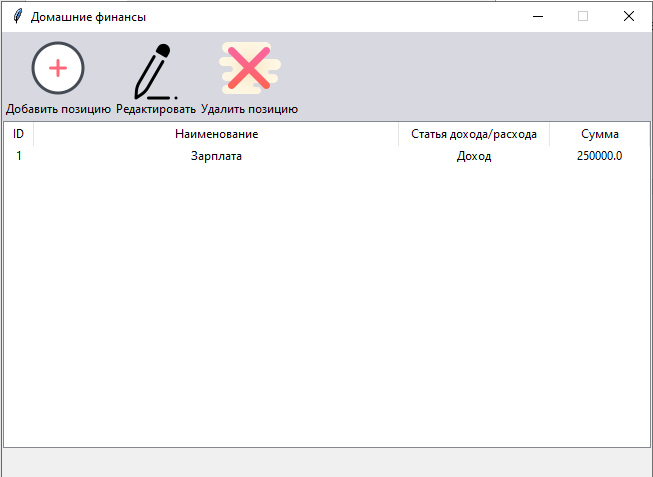


Для добавления данных нажимаем кнопку «Добавить позицию».

Открывается дочернее окно с формой ввода. В форму ввода мы вписываем данные, которые хотим добавить и нажимаем «добавить».

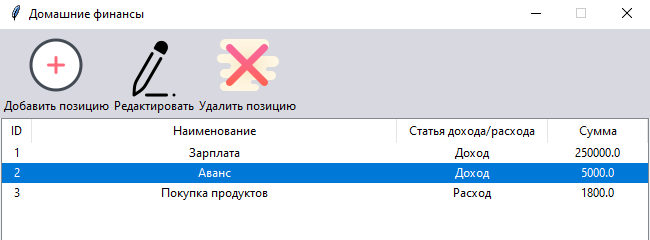


Данные, которые были введены автоматически добавляются в базу данных и отображаются в главном окне.

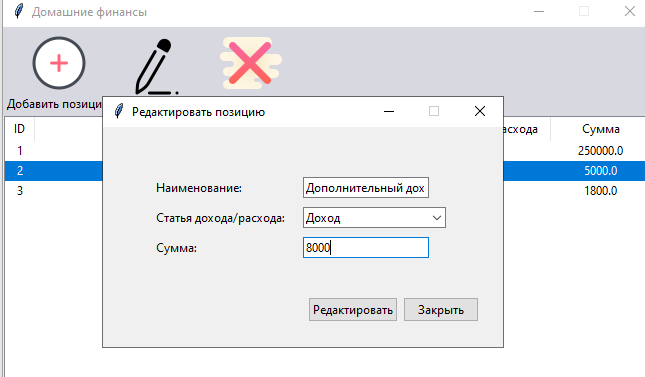


# 2.2. Редактирование данных

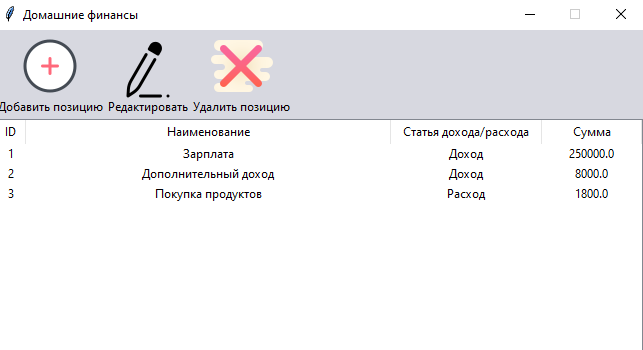
Для редактирования выделяем нужный пункт чтобы обновить данные. И нажимаем кнопку «Редактировать».



Появляется форма для добавления данных. Вводим в нее новые данные



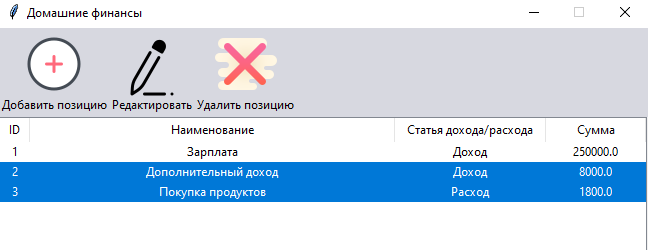
И нажимает редактировать данные автоматически обновляются.



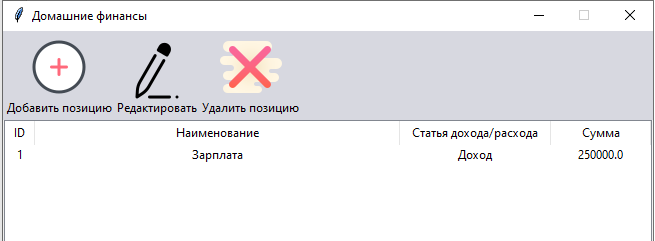
Таким образом редактирование завершается.

# Удаление данных

Для удаления данных выбираем один или несколько пунктов данных



И нажимаем «Удалить позицию». Видим, что данные, которые мы выделили удалились.



ЗАКЛЮЧЕНИЕ

Передо мной стояла следующая задача:

* Создание приложения с базой данных и графическим интерфейсом
* Создайте необходимые запросы в базу. Оформите их в виде отдельных функций.
* Свяжите созданные функции с элементами управления. протестируйте программу.

Все поставленные задачи были выполнены, а именно:

* Создана программа с базой данных и графическим интерфейсом
* Созданы запросы в базу такие как «Добавить» «Редактировать» «Удалить».
* Созданные функции все связаны с элементами управления.

# ПРИЛОЖЕНИЕ 1

# 

Главное окно

Класс

Main()

Добавление позиции

btn\_open\_dialog

Редактирование позиции

btn\_edit\_dialog

Удаление позиции

btn\_delete

Дочернее окно

Класс

Child(0

Класс базы данных

BD()

Схема взаимодействия модулей программы