Кафедра систем управления и вычислительной техники

Работа защищена с оценкой

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

(дата)

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

(подпись и Ф.И.О. преподавателя)

ПРОГРАММИРОВАНИЕ

Пояснительная записка по курсовой работе

Работу выполнил студент

учебной группы \_\_\_\_\_\_\_

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

(подпись и Ф.И.О. студента)

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

(дата)

Работа допущена

к защите

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

(подпись и Ф.И.О.

руководителя работы)

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

(дата)

# Содержание

[1. Тема 24. «Обработка прерываний» 3](#_Toc42037814)

[1.1. Теоретическое описание темы «Обработка прерываний» 3](#_Toc42037815)

[1.2. Примеры задач и программ по теме «Обработка прерываний» 6](#_Toc42037816)

[1.3. Список использованных источников 8](#_Toc42037817)

[2.1. Постановка задачи. Вариант №7 9](#_Toc42037818)

[2.2. Описание используемых в программе классов 11](#_Toc42037819)

[2.3. Макет GUI c описанием использованных виджетов 12](#_Toc42037820)

[2.4. Описание событий и методов с ними связанных 13](#_Toc42037821)

[2.5. Текст Программы 13](#_Toc42037822)

[2.6. Текст исходного файла 18](#_Toc42037823)

[2.7. Скриншоты выполнения программы 19](#_Toc42037824)

# Тема 24. «Обработка прерываний»

# Теоретическое описание темы «Обработка прерываний»

**Прерывание** (англ. interrupt) — сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-либо события, требующего немедленного внимания. Прерывание извещает процессор о наступлении высокоприоритетного события, требующего прерывания текущего кода, выполняемого процессором. Процессор отвечает приостановкой своей текущей активности, сохраняя свое состояние и выполняя функцию, называемую обработчиком прерывания (или программой обработки прерывания), которая реагирует на событие и обслуживает его, после чего возвращает управление в прерванный код.

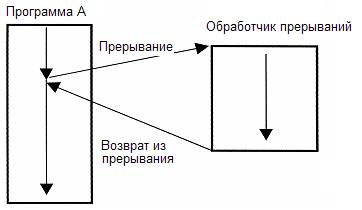
**Обработчик прерываний** – программа обработки прерывания, являющаяся частью ОС, предназначенная для выполнения ответных действий на условие, вызвавшее прерывание.

Прерывания бывают двух видов:

1. **Аппаратные прерывания**

используются устройствами для передачи информации о том, что они требуют внимания со стороны операционной системы.

1. **Программные прерывания**

вызвано либо исключительным состоянием в самом процессоре, либо специальной инструкцией в наборе команд, которая вызывает прерывание, когда инструкция выполняется (см. рисунок 1). 

*(рис. 1)*

Программные прерывания процессором обрабатываются как разновидность исключений.

**Обрабо́тка исключи́тельных ситуа́ций** (англ. exception handling) — механизм языков программирования, предназначенный для описания реакции программы на ошибки времени выполнения и другие возможные проблемы (исключения), которые могут возникнуть при выполнении программы и приводят к невозможности (бессмысленности) дальнейшей отработки программой её базового алгоритма. В русском языке также применяется более короткая форма термина: «обработка исключений»

**Список исключений (python 3.3)**

|  |  |
| --- | --- |
| **Кодовое название** | **Описание** |
| BaseException | базовое исключение, от которого берут начало все остальные. |
| * SystemExit | исключение, порождаемое функцией sys.exit при выходе из программы. |
| * KeyboardInterrupt | порождается при прерывании программы пользователем (обычно сочетанием клавиш Ctrl+C). |
| * GeneratorExit | порождается при вызове метода close объекта generator. |
| * Exception | а вот тут уже заканчиваются полностью системные исключения и начинаются обыкновенные. |
| * + StopIteration | порождается встроенной функцией next, если в итераторе больше нет элементов. |
| * + ArithmeticError | арифметическая ошибка. |
| * + - FloatingPointError | порождается при неудачном выполнении операции с плавающей запятой. На практике встречается нечасто. |
| * + - OverflowError | возникает, когда результат арифметической операции слишком велик для представления. Не появляется при обычной работе с целыми числами (так как python поддерживает длинные числа), но может возникать в некоторых других случаях. |
| * + - ZeroDivisionError | деление на ноль. |
| * + AssertionError | выражение в функции assert ложно. |
| * + AttributeError | объект не имеет данного атрибута (значения или метода). |
| * + BufferError | операция, связанная с буфером, не может быть выполнена. |
| * + EOFError | функция наткнулась на конец файла и не смогла прочитать то, что хотела. |
| * + ImportError | не удалось импортирование модуля или его атрибута. |
| * + LookupError | некорректный индекс или ключ. |
| * + - IndexError | индекс не входит в диапазон элементов. |
| * + - KeyError | несуществующий ключ (в словаре, множестве или другом объекте). |
| * + MemoryError | недостаточно памяти. |
| * + NameError | не найдено переменной с таким именем. |
| * + - UnboundLocalError | сделана ссылка на локальную переменную в функции, но переменная не определена ранее. |
| * + OSError | ошибка, связанная с системой. |
| * + - BlockingIOError |  |
| * + - ChildProcessError | неудача при операции с дочерним процессом. |
| * + - ConnectionError | базовый класс для исключений, связанных с подключениями. |
| * + - * BrokenPipeError |  |
| * + - * ConnectionAbortedError |  |
| * + - * ConnectionRefusedError |  |
| * + - * ConnectionResetError |  |
| * + - FileExistsError | попытка создания файла или директории, которая уже существует. |
| * + - FileNotFoundError | файл или директория не существует. |
| * + - InterruptedError | системный вызов прерван входящим сигналом. |
| * + - IsADirectoryError | ожидался файл, но это директория. |
| * + - NotADirectoryError | ожидалась директория, но это файл. |
| * + - PermissionError | не хватает прав доступа. |
| * + - ProcessLookupError | указанного процесса не существует. |
| * + - TimeoutError | закончилось время ожидания. |
| * + ReferenceError | попытка доступа к атрибуту со слабой ссылкой. |
| * + RuntimeError | возникает, когда исключение не попадает ни под одну из других категорий. |
| * + NotImplementedError | возникает, когда абстрактные методы класса требуют переопределения в дочерних классах. |
| * + SyntaxError | синтаксическая ошибка. |
| * + - IndentationError | неправильные отступы. |
| * + - * TabError | смешивание в отступах табуляции и пробелов. |
| * + SystemError | внутренняя ошибка. |
| * + TypeError | операция применена к объекту несоответствующего типа. |
| * + ValueError | функция получает аргумент правильного типа, но некорректного значения. |
| * + UnicodeError | ошибка, связанная с кодированием / раскодированием unicode в строках. |
| * + - UnicodeEncodeError | исключение, связанное с кодированием unicode. |
| * + - UnicodeDecodeError | исключение, связанное с декодированием unicode. |
| * + - UnicodeTranslateError | исключение, связанное с переводом unicode. |
| * + Warning | предупреждение. |

# Примеры задач и программ по теме «Обработка прерываний»

В Python ошибки называются исключения (exceptions).

Примеры исключений:

4/0

-----------------------------------------------------

ZeroDivisionError: division by zero

'test' + 2

-----------------------------------------------------

TypeError: must be str, not int

В данном случае возникло два исключения: ZeroDivisionError и TypeError.

Можно писать программы, которые обрабатывают перечисленные выше исключения. Самый простой способ обработки исключений - с помощью блока try-except:

x = 4

y = 0

try:

z = x/y

except ZeroDivisionError:

print "Невозможно делить на ноль"

Следующий пример запрашивает ввод у пользователя до тех пор, пока не будет введено правильное целое число, но позволяет пользователю прерывать программу (используя Control-C или все, что поддерживает операционная система);

while True:

try:

x = int(input("Пожалуйста, введите целое число:"))

break

except ValueError:

print("Упс! Произошла ошибка. Попробуйте ещё раз...")

Оператор try-except имеет необязательное продолжение else, которое, если оно присутствует, должно быть последним. Это полезно для кода, который должен быть выполнен, если предложение try не вызывает исключение. Например:

for arg in sys.argv[1:]:

try:

f = open(arg, 'r')

except OSError:

print('Невозможно открыть:', arg)

else:

print(arg, 'количество строк -', len(f.readlines()))

f.close()

Finally выполняет блок инструкций в любом случае, было ли исключение, или нет (применима, когда нужно непременно что-то сделать, к примеру, закрыть файл). Инструкция else выполняется в том случае, если исключения не было.

file = open(“numbers.txt”)

numbers = []

try:

for line in file:

numbers.append(int(line))

except ValueError:

print(“Строка не является числом.”)

except Exception:

print(“Произошла ошибка.”)

else:

print(“Успешно!”)

finally:

print(“Завершение операции…”)

file.close()

Таким образом, данная программа будет выполнять операции в таком порядке:

try -> except(2 случая) -> else -> finally

# Список использованных источников

1. <https://docs.python.org/3/>
2. Intel 64 and IA-32 Architectures Software Developer’s Manual (в пяти томах):
   * [Том 3A. System Programming Guide. CHAPTER 6. INTERRUPT AND EXCEPTION HANDLING](https://web.archive.org/web/20081221050950/http:/download.intel.com/design/processor/manuals/253668.pdf) (англ.)
3. The Quick Python Book, Third Edition. Chapter 14. Exceptions
   * <https://livebook.manning.com/book/the-quick-python-book-third-edition/chapter-14/>

# Постановка задачи. Вариант №7

* Наименование задачи – получение сведений об автомобилях
* Структура записей исходного файла:
  + Марка автомобиля
  + Страна
  + Год выпуска
  + Стоимость
* Запрос – год выпуска
* Результаты:
  + Список выпущенных

автомобилей не ранее

введенного года с указанием

марки, страны и стоимости.

* + Количество автомобилей

Написание оригинальной программы согласно варианту, выданному

студенту преподавателем. Выполнение задачи должно быть оформлено

графическим интерфейсом. При разработке формы GUI использовать

следующие виджеты:

* метка с указанием имени и фамилии студента, а также его группы;
* кнопки начала и завершения работы;
* кнопки для поиска информации и расчетов;
* однострочное тестовое поле для ввода имени файла;
* однострочное тестовое поле для ввода условия отбора записей из файла;
* метки для вывода результатов расчета;
* список или многострочное тестовое поле для вывода данных файла, удовлетворяющих условию.

Дизайн формы студент определяет самостоятельно.

Информация хранится в файле (10-15 записей), структура записи файла

определена вариантом задания. Имя файла вводится в окно ввода, по кнопке

«Открыть» файл открывается. В поле вводится условие поиска информации,

результат – в поле список или многострочное тестовое поле записей,

удовлетворяющие заданному условию. Кнопка «Расчет» инициирует расчеты

и вывод результата. По числовым данным строится график. Кнопка

«Закрыть» закрывает файл и форму.

# Описание используемых в программе классов

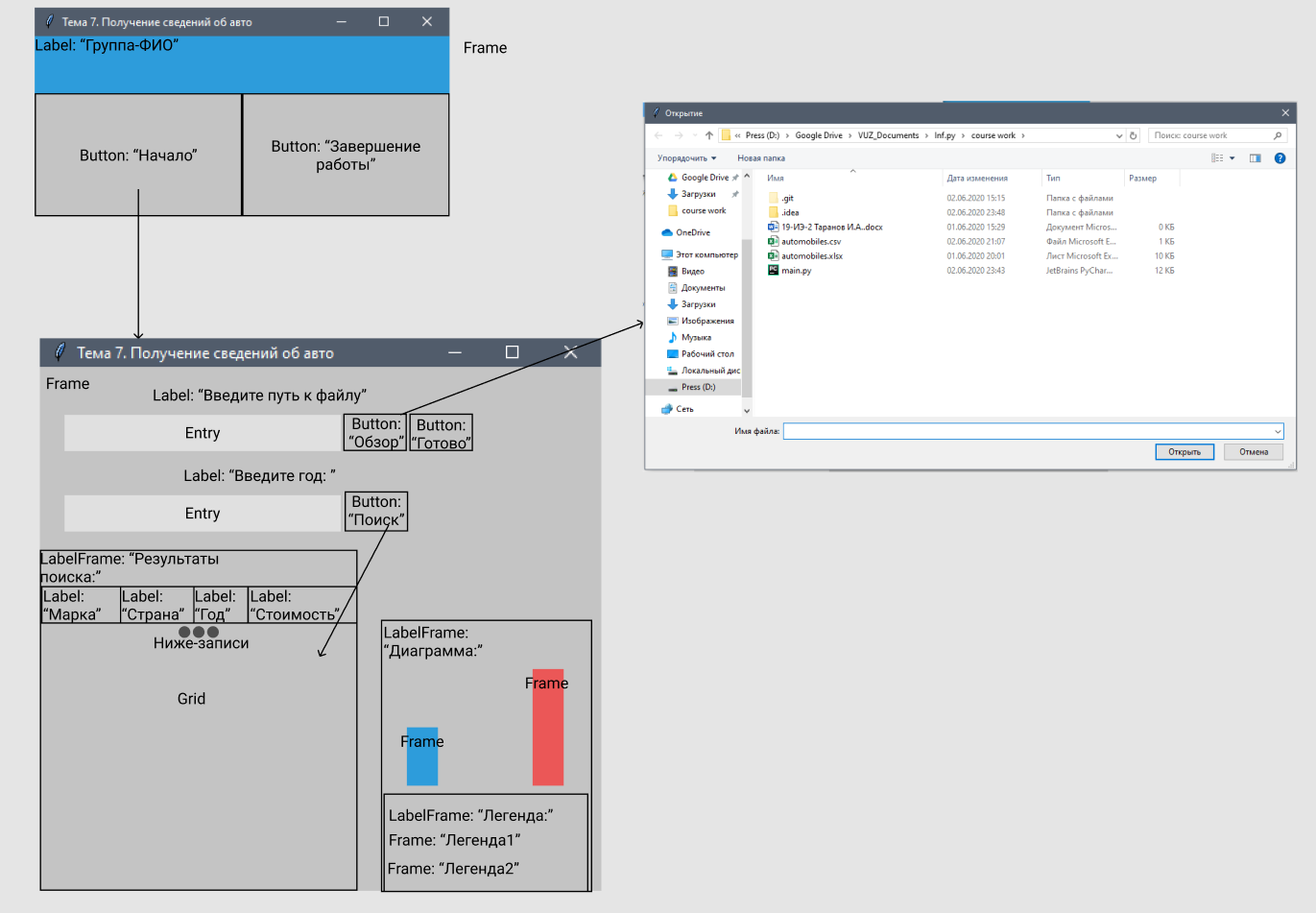
В программе используется один класс – Car. Название само говорит за себя, класс олицетворяет сущность записи машины. Класс имеет 4 поля, 1 метод - «не старше, чем год».

Карточка класса:

Автомобиль

* Марка (str)
* Страна (str)
* Год выпуска (int)
* Стоимость (int)
* НеСтаршеЧем(год)

# Макет GUI c описанием использованных виджетов



https://www.figma.com/file/o7HcC9Qmoh8sN0hlDFLtJv/%D0%9A%D1%83%D1%80%D1%81%D0%BE%D0%B2%D0%B0%D1%8F?node-id=0%3A1

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип** | **Название** | **Код** | **Описание** |
| Label | Группа-ФИО | student\_text\_label | Информация о студенте |
| Button | Начало | start\_button | Запускает главное окно, старт программы |
| Button | Завершение работы | close\_button | Завершает программу |
| Button | Обзор | search\_file\_button | Запускает проводник для выбора таблицы |
| Entry |  | input\_file\_text\_label | Хранится запись пути файла. Вставляется текст после нажатия «открыть» |
| Button | Готово | accept\_file\_button | Подтверждение использования пути к таблице |
| Label | Введите год: | input\_year\_text\_label | Пояснение к действию |
| Entry |  | input\_year\_frame | Поле ввода для поиска |
| Button | Поиск | search\_button | Выполняет функцию поиска |
| LabelFrame | Результаты поиска | search\_result\_frame | Поле с результатами |
|  | Машина | Car | Label создаются циклично, заполняя Grid |
| * Label | Марка | model |  |
| * Label | Страна | country |  |
| * Label | Год | year |  |
| * Label | Стоимость | cost |  |
| LabelFrame | Диаграмма | diagram\_frame | Поле с диаграммой |
| Frame |  | legend\_not\_qualified\_frame | Диаграмма числа машин, не соответствующих поиску |
| Frame |  | legend\_qualified\_frame | Диаграмма числа машин, соответствующих поиску |
| LabelFrame | Легенда | diagram\_legend\_frame | Поле с легендой |

# Описание событий и методов с ними связанных

Код программы можно назвать самодокументируемым, поскольку он достаточно пояснён комментариями и, в целом, понятен.

Таблица событий

|  |  |  |
| --- | --- | --- |
| **Описание события** | **Код** | **Результат** |
| Нажатие кнопки «начало» | start\_program() | Запуск главного окна – основной программы |
| Нажатие кнопки «завершение программы» | close\_program() | Закрытие программы и всех окон |
| Нажатие кнопки «обзор» | search\_file(): | Запуск проводника, где пользователь вводит путь к файлу таблицы с данными |
| Ввод вручную в поле Entry | input\_file\_entry | Помимо использования проводника, можно ввести путь к файлу вручную в поле |
| Нажатие кнопки «готово» | accept\_file() | Подтверждает использование файла, будь то ввод вручную, или с помощью проводника |
| Ввод в поле «введите год» и нажатие кнопки «готово» | search\_cars(frame, year) | Осуществляет поиск, выводит результат в виде таблице в поле ниже |

# Текст Программы

<https://github.com/ivatar39/first_course_work>

# Импорт библиотеки tkinter для работы с GUI  
from tkinter import \*  
from tkinter import filedialog as fd  
  
  
# Класс машины  
class Car:  
 # Конструктор, принимающий в себя необходимые поля  
 def \_\_init\_\_(*self*, model, country, year, cost):  
 *self*.model = model  
 *self*.country = country  
 *self*.year = int(year)  
 *self*.cost = cost  
  
 # Метод, возвращающий true если год выпуска не ранее срока  
 def not\_older(*self*, year):  
 return *self*.year >= year  
  
  
# Функция, которая считывает таблицу с данными  
def read\_database(filename):  
 try:  
 file = open(filename, encoding=**'UTF-8'**)  
 file\_lines = file.readlines()  
 # Удаление заголовков  
 file\_lines.pop(0)  
 processed\_lines = []  
 # Форматирование csv-файла  
 for line in file\_lines:  
 processed\_lines.append(line.replace(**'**\n**'**, **''**).rsplit(**';'**))  
 file.close()  
 return processed\_lines  
 except FileNotFoundError:  
 print(**'Файл не найден'**)  
  
  
# Функция, которая принимает список и возвращает список объектов-машин  
def convert\_lines\_to\_cars(lines):  
 convert\_cars = []  
 for line in lines:  
 convert\_cars.append(Car(model=line[0], country=line[1], year=line[2], cost=line[3]))  
 return convert\_cars  
  
  
# Функция, которая принимает список машин и возвращает список машин, не ранее указанного года  
def get\_cars\_by\_year(search\_cars\_by\_year, year):  
 filtered\_cars = []  
 for search\_car in search\_cars\_by\_year:  
 if search\_car.not\_older(year):  
 filtered\_cars.append(search\_car)  
 return filtered\_cars  
  
  
# Функция закрытия программы  
def close\_program():  
 exit()  
  
  
# Функция, осуществляющая отрисовку поиска (рисует рез. поиска, либо отсутствие рез.)  
def search\_cars(search\_frame, year):  
 # Очистка виджетов перед отрисовкой  
 for children in search\_frame.winfo\_children():  
 children.destroy()  
 try:  
 # Получаем список машин, соответсвующих поисковому запросу  
 searched\_cars = get\_cars\_by\_year(cars, int(year))  
 if len(searched\_cars) > 0:  
 draw\_cars\_table(search\_frame, searched\_cars)  
 # Считаем количество найденных и не найденных машин  
 cars\_qualified = len(searched\_cars)  
 cars\_not\_qualified = len(cars) - cars\_qualified  
 # Рисуем диаграмму  
 draw\_diagram(search\_frame, cars\_not\_qualified, cars\_qualified)  
 # Если поиск не дал результатов, возварщаем соответсвующее окно  
 else:  
 draw\_no\_result(search\_frame)  
 except ValueError:  
 draw\_no\_result(search\_frame)  
  
  
# Отрисовка таблицы - результат поискового запроса  
def draw\_cars\_table(search\_frame, draw\_cars):  
 search\_result\_frame = LabelFrame(search\_frame, height=250, width=100, text=**'Результаты поиска:'**)  
 search\_result\_frame.pack(side=LEFT)  
  
 # Верхняя строка - заголовки  
 model\_key\_label = Label(search\_result\_frame, text=**'Марка автомобиля'**)  
 model\_key\_label.grid(row=0, column=0)  
 country\_key\_label = Label(search\_result\_frame, text=**'Страна'**)  
 country\_key\_label.grid(row=0, column=1)  
 year\_key\_label = Label(search\_result\_frame, text=**'Год'**)  
 year\_key\_label.grid(row=0, column=2)  
 price\_key\_label = Label(search\_result\_frame, text=**'Стоимость'**)  
 price\_key\_label.grid(row=0, column=3)  
  
 # Для каждой машины отрисовываем поля с данными  
 for index in range(len(draw\_cars)):  
 car\_model\_label = Label(search\_result\_frame, text=draw\_cars[index].model)  
 car\_model\_label.grid(row=index+1, column=0)  
 car\_country\_label = Label(search\_result\_frame, text=draw\_cars[index].country)  
 car\_country\_label.grid(row=index + 1, column=1)  
 car\_year\_label = Label(search\_result\_frame, text=draw\_cars[index].year)  
 car\_year\_label.grid(row=index + 1, column=2)  
 car\_cost\_label = Label(search\_result\_frame, text=draw\_cars[index].cost)  
 car\_cost\_label.grid(row=index + 1, column=3)  
  
  
# Функция, отрисовывающая окно, когда поиск не дал результатов  
def draw\_no\_result(search\_frame):  
 search\_result\_frame = Frame(search\_frame)  
 search\_result\_frame.pack(side=BOTTOM)  
 no\_result\_label = Label(search\_result\_frame, text=**'Поиск не дал результатов'**)  
 no\_result\_label.pack(padx=100, pady=100)  
  
  
# Функция отрисовки диаграммы - кол-во найденных авто  
def draw\_diagram(search\_frame, num\_cars\_search\_not\_qualified, num\_cars\_search\_qualified):  
 diagram\_frame = LabelFrame(search\_frame, width=30, text=**'Диаграмма'**)  
 diagram\_frame.pack(side=RIGHT, padx=14)  
 max\_height = 10  
 not\_qualified\_color = **'dodger blue'** qualified\_color = **'firebrick1'** # Отрисовка легенды  
 diagram\_legend\_frame = LabelFrame(diagram\_frame, text=**'Легенда'**)  
 diagram\_legend\_frame.pack(side=BOTTOM, padx=4, pady=4)  
 # Отрисовка легенды "не соотв. поиску"  
 legend\_not\_qualified\_frame = Frame(diagram\_legend\_frame)  
 legend\_not\_qualified\_color\_frame = Frame(legend\_not\_qualified\_frame, bg=not\_qualified\_color, width=5, height=5)  
 legend\_not\_qualified\_color\_frame.pack(side=LEFT)  
 legend\_not\_qualified\_label = Label(legend\_not\_qualified\_frame, text=**'Не соответствует результатам поиска'**)  
 legend\_not\_qualified\_label.pack(side=RIGHT)  
 legend\_not\_qualified\_frame.pack(side=TOP, padx=1, pady=1, anchor=NW)  
 # Отрисовка легенды "соотв. поиску"  
 legend\_qualified\_frame = Frame(diagram\_legend\_frame)  
 legend\_qualified\_color\_frame = Frame(legend\_qualified\_frame, bg=qualified\_color, width=5, height=5)  
 legend\_qualified\_color\_frame.pack(side=LEFT)  
 legend\_qualified\_label = Label(legend\_qualified\_frame, text=**'Соответствует результатам поиска'**)  
 legend\_qualified\_label.pack(side=RIGHT)  
 legend\_qualified\_frame.pack(side=BOTTOM, padx=1, pady=1, anchor=SW)  
  
 # Спец. случай если поисковый запрос вернул все авто  
 if num\_cars\_search\_not\_qualified == 0:  
 qualified\_diagram = Label(diagram\_frame, width=1, height=max\_height, bg=qualified\_color,  
 text=num\_cars\_search\_qualified)  
 qualified\_diagram.pack(side=RIGHT, anchor=SE, padx=5)  
 # Спец. случай если поисковый запрос вернул 0 авто  
 elif num\_cars\_search\_qualified == 0:  
 not\_qualified\_diagram = Label(diagram\_frame, width=1, height=max\_height, bg=not\_qualified\_color,  
 text=num\_cars\_search\_not\_qualified)  
 not\_qualified\_diagram.pack(side=LEFT, anchor=SW, padx=5)  
 # Иначе - с помощью коэффициента высчитываем соотношение кол-во найденных и не найденных авто  
 else:  
 num\_overall = num\_cars\_search\_not\_qualified + num\_cars\_search\_qualified  
 not\_qualified\_height = (num\_cars\_search\_not\_qualified / num\_overall) \* max\_height  
 qualified\_height = (num\_cars\_search\_qualified / num\_overall) \* max\_height  
  
 not\_qualified\_diagram = Label(diagram\_frame, width=1, height=int(not\_qualified\_height), bg=not\_qualified\_color,  
 text=num\_cars\_search\_not\_qualified)  
 not\_qualified\_diagram.pack(side=LEFT, anchor=SW, padx=5)  
  
 qualified\_diagram = Label(diagram\_frame, width=1, height=int(qualified\_height), bg=qualified\_color,  
 text=num\_cars\_search\_qualified)  
 qualified\_diagram.pack(side=RIGHT, anchor=SE, padx=5)  
  
  
# Запуск главной программы  
def start\_program():  
 # Программа  
 def accept\_file():  
 file\_name = input\_file\_entry.get()  
 global cars  
 cars = convert\_lines\_to\_cars(read\_database(file\_name))  
  
 # Поиск пути в проводнике  
 def search\_file():  
 file\_name = fd.askopenfilename()  
 # Замена поля ввода  
 input\_file\_entry.delete(0, END)  
 input\_file\_entry.insert(0, file\_name)  
  
 global cars  
 cars = convert\_lines\_to\_cars(read\_database(file\_name))  
  
 main\_window = Toplevel(root)  
 main\_window.minsize(width=400, height=300)  
 # Главный фрэйм приложения  
 app\_frame = Frame(main\_window)  
 app\_frame.pack()  
 # Фрейм ввода файла  
 input\_file\_frame = Frame(app\_frame)  
 input\_file\_frame.pack(side=TOP, pady=10)  
 input\_file\_text\_label = Label(input\_file\_frame, text=**'Введите путь к файлу:'**)  
 input\_file\_text\_label.pack(side=TOP)  
 input\_file\_entry = Entry(input\_file\_frame, width=100)  
 input\_file\_entry.pack(side=LEFT)  
 accept\_file\_button = Button(input\_file\_frame, text=**'Готово'**, height=1,  
 command=lambda: accept\_file())  
 accept\_file\_button.pack(side=RIGHT)  
 search\_file\_button = Button(input\_file\_frame, text=**'Обзор'**, height=1,  
 command=lambda: search\_file())  
 search\_file\_button.pack(side=RIGHT)  
  
 # Фрейм для результатов поиска  
 search\_place\_frame = Frame(app\_frame, height=100)  
 search\_place\_frame.pack(side=BOTTOM, padx=30, pady=30)  
  
 # Фрейм ввода года  
 input\_year\_frame = Frame(app\_frame)  
 input\_year\_frame.pack(side=TOP, pady=10)  
 input\_year\_text\_label = Label(input\_year\_frame, text=**'Введите год:'**)  
 input\_year\_text\_label.pack(side=TOP)  
 input\_year\_entry = Entry(input\_year\_frame, width=30)  
 input\_year\_entry.pack(side=LEFT)  
 search\_button = Button(input\_year\_frame, text=**'Поиск'**, height=1,  
 command=lambda: search\_cars(search\_place\_frame, input\_year\_entry.get()))  
 search\_button.pack(side=RIGHT)  
  
  
# Главное Приложение, запуск стартового экрана  
root = Tk()  
cars = []  
  
root.title(**'Тема 7. Получение сведений об авто'**)  
# Фрэйм-шапка приложения  
app\_bar\_frame = Frame(root, bg=**'SteelBlue1'**)  
app\_bar\_frame.pack(side=TOP)  
close\_button = Button(app\_bar\_frame, text=**'Завершение работы'**, width=30, height=5, command=lambda: close\_program())  
close\_button.pack(side=RIGHT, expand=1, anchor=SE)  
student\_text\_label = Label(app\_bar\_frame, text=**'19-ИЭ-2 Таранов И.А.'**, bg=**'SteelBlue1'**)  
student\_text\_label.pack(side=TOP)  
start\_button = Button(app\_bar\_frame, text=**'Начало'**, width=30, height=5, command=lambda: start\_program())  
start\_button.pack(side=LEFT, expand=1, anchor=SW)  
  
# Главный цикл  
root.mainloop()

# Текст исходного файла

|  |  |  |  |
| --- | --- | --- | --- |
| марка автомобиля | страна | год выпуска | стоимость |
| Toyota | Япония | 2012 | 1 800 000,00 ₽ |
| Toyota | Япония | 2000 | 800 000,00 ₽ |
| Volkswagen | Германия | 2012 | 2 471 000,00 ₽ |
| Volkswagen | Германия | 2011 | 1 174 000,00 ₽ |
| Tata Motors | Монголия | 2010 | 713 000,00 ₽ |
| Tata Motors | Монголия | 2007 | 410 000,00 ₽ |
| Honda | Англия | 2010 | 6 460 000,00 ₽ |
| Honda | Англия | 2005 | 3 230 000,00 ₽ |
| Tesla | США | 2016 | 10 000 000,00 ₽ |
| Tesla | США | 2019 | 25 700 000,00 ₽ |
| Porsche | Италия | 2001 | 15 000 000,00 ₽ |
| Lada | Россия | 2007 | 309 999,00 ₽ |
| Lada | Россия | 2000 | 1 213 321,00 ₽ |
| Porsche | Италия | 2001 | 7 500 000,00 ₽ |
| Tesla | США | 2020 | 30 000 000,00 ₽ |

# Скриншоты выполнения программы

