Листинг программы

main.py

**import** sys  
**import** tkinter **as** tk  
**import** tkinter.font **as** tkfont  
**import** tkinter.ttk **as** ttk  
**from** tkinter **import** IntVar  
**from** tkinter.messagebox **import** showinfo  
**import** pandas **as** pd  
  
**from** params **import** \*  
**from** reportmodule **import** \*  
  
sys.path.append(LIBRARY\_PATH)  
**import** databin  
  
  
**def** sortby(tree, col, descending):  
 *"""  
 Функция для сортировки строк таблицы  
 Параметры: tree - TreeView, col - столбец, descending - порядок сортировки  
 Автор: Марков Д.Э.  
 """* **def** change\_numeric(data):  
 *"""  
 Функция для приведения элементов массива к типу int  
 Параметры: data - массив  
 Возвращает новый массив  
 Автор: Марков Д.Э.  
 """* **return** [(int(x[0]), x[1]) **for** x **in** data]  
  
 data = [(tree.set(child, col), child) **for** child **in** tree.get\_children(**''**)]  
 **if** data[0][0].isdigit():  
 data = change\_numeric(data)  
 data.sort(reverse=descending)  
 **for** ix, item **in** enumerate(data):  
 tree.move(item[1], **''**, ix)  
 tree.heading(col, command=**lambda** col=col: sortby(tree, col, int(**not** descending)))  
  
  
**def** open\_edit\_window():  
 *"""  
 Функция для открытия окна редактирования  
 Автор: Марков Д.Э.  
 """* **global** df  
  
 **def** edit\_entry():  
 *"""  
 Функция для редактирования выбранной записи базы данных  
 Автор: Марков Д.Э.  
 """* **if** selected\_database.get() == 1:  
 keys = [**'Название'**, **'Дата выхода'**, **'Конфигурация памяти, ГБ'**, **'Энергопотребление, Вт'**, **'Far Cry 5, FPS'**,  
 **'Fallout 4, FPS'**, **'The Witcher 3, FPS'**, **'3DMark Cloud Gate'**, **'3DMark Fire Strike'**,  
 **'Средняя цена, ₽'**, **'Архитектура'**, **'NVIDIA SLI'**, **'RTX'**, **'Базовая тактовая частота, МГц'**]  
 values = [  
 entry\_name.get(),  
 entry\_date.get(),  
 entry\_memory.get(),  
 entry\_power.get(),  
 entry\_farcry5.get(),  
 entry\_fallout4.get(),  
 entry\_thewitcher3.get(),  
 entry\_cloudgate.get(),  
 entry\_firestrike.get(),  
 entry\_price.get(),  
 entry\_arch.get(),  
 entry\_sli.get(),  
 entry\_rtx.get(),  
 entry\_freq.get()  
 ]  
 **if "" in** values:  
 showinfo(ERROR\_MESSAGE, **"Заполнены не все поля"**)  
 **else**:  
 entry = dict(zip([x **for** x **in** keys], [y **for** y **in** values]))  
 df.loc[(entry[**'Название'**], int(entry[**'Конфигурация памяти, ГБ'**])), list(df.columns)] = [entry[x] **for** x  
 **in** df.columns]  
 **elif** selected\_database.get() == 2:  
 values = [entry\_name.get(), entry\_arch.get()]  
 **if "" in** values:  
 showinfo(ERROR\_MESSAGE, **"Заполнены не все поля"**)  
 **else**:  
 **for** index **in** df.index:  
 **if** df.loc[index][**'Название'**] == key:  
 df.loc[index, **'Архитектура'**] = values[1]  
 **elif** selected\_database.get() == 3:  
 values = [entry\_name.get(), entry\_sli.get()]  
 **if "" in** values:  
 showinfo(ERROR\_MESSAGE, **"Заполнены не все поля"**)  
 **else**:  
 **for** index **in** df.index:  
 **if** df.loc[index][**'Название'**] == key:  
 df.loc[index, **'NVIDIA SLI'**] = values[1]  
 **elif** selected\_database.get() == 4:  
 values = [entry\_name.get(), entry\_rtx.get()]  
 **if "" in** values:  
 showinfo(ERROR\_MESSAGE, **"Заполнены не все поля"**)  
 **else**:  
 **for** index **in** df.index:  
 **if** df.loc[index][**'Название'**] == key:  
 df.loc[index, **'RTX'**] = values[1]  
 **elif** selected\_database.get() == 5:  
 values = [entry\_name.get(), entry\_freq.get()]  
 **if "" in** values:  
 showinfo(ERROR\_MESSAGE, **"Заполнены не все поля"**)  
 **else**:  
 **for** index **in** df.index:  
 **if** df.loc[index][**'Название'**] == key:  
 df.loc[index, **'Базовая тактовая частота, МГц'**] = values[1]  
  
 update\_table()  
 window\_edit.destroy()  
  
 entry = tree.focus()  
 **if** len(entry) != 0:  
 window\_edit = tk.Toplevel(window)  
 window\_edit.resizable(0, 0)  
  
 **if** selected\_database.get() == 1:  
 **if '{' in** entry:  
 y = entry.split(**'}'**)  
 k1 = y[0][1:]  
 x2 = y[1].split(**' '**)  
 k2 = int(float(x2[2]))  
 **else**:  
 y = entry.split(**' '**)  
 k1 = y[0]  
 k2 = y[2]  
 key = (k1, int(k2))  
 tk.Label(window\_edit, text=**'Редактировать запись'**).grid(row=0, columnspan=2)  
 tk.Label(window\_edit, text=**'Имя:'**).grid(row=1, column=0)  
 tk.Label(window\_edit, text=**'Дата выхода:'**).grid(row=2, column=0)  
 tk.Label(window\_edit, text=**'Конфигурация памяти, ГБ:'**).grid(row=3, column=0)  
 tk.Label(window\_edit, text=**'Энергопотребление'**).grid(row=4, column=0)  
 tk.Label(window\_edit, text=**'Far Cry 5, FPS'**).grid(row=5, column=0)  
 tk.Label(window\_edit, text=**'Fallout 4, FPS'**).grid(row=6, column=0)  
 tk.Label(window\_edit, text=**'The Witcher 3, FPS'**).grid(row=7, column=0)  
 tk.Label(window\_edit, text=**'3DMark Cloud Gate'**).grid(row=8, column=0)  
 tk.Label(window\_edit, text=**'3DMark Fire Strike'**).grid(row=9, column=0)  
 tk.Label(window\_edit, text=**'Средняя цена, руб.'**).grid(row=10, column=0)  
 tk.Label(window\_edit, text=**'Архитектура'**).grid(row=11, column=0)  
 tk.Label(window\_edit, text=**'NVIDIA SLI'**).grid(row=12, column=0)  
 tk.Label(window\_edit, text=**'RTX'**).grid(row=13, column=0)  
 tk.Label(window\_edit, text=**'Базовая тактовая частота'**).grid(row=14, column=0)  
  
 entry\_name = tk.Entry(window\_edit)  
 entry\_name.insert(0, df.loc[key][**'Название'**])  
 entry\_name.grid(row=1, column=1)  
 entry\_date = tk.Entry(window\_edit)  
 entry\_date.insert(0, df.loc[key][**'Дата выхода'**])  
 entry\_date.grid(row=2, column=1)  
 entry\_memory = tk.Entry(window\_edit)  
 entry\_memory.insert(0, df.loc[key][**'Конфигурация памяти, ГБ'**])  
 entry\_memory.grid(row=3, column=1)  
 entry\_power = tk.Entry(window\_edit)  
 entry\_power.insert(0, df.loc[key][**'Энергопотребление, Вт'**])  
 entry\_power.grid(row=4, column=1)  
 entry\_farcry5 = tk.Entry(window\_edit)  
 entry\_farcry5.insert(0, df.loc[key][**'Far Cry 5, FPS'**])  
 entry\_farcry5.grid(row=5, column=1)  
 entry\_fallout4 = tk.Entry(window\_edit)  
 entry\_fallout4.insert(0, df.loc[key][**'Fallout 4, FPS'**])  
 entry\_fallout4.grid(row=6, column=1)  
 entry\_thewitcher3 = tk.Entry(window\_edit)  
 entry\_thewitcher3.insert(0, df.loc[key][**'The Witcher 3, FPS'**])  
 entry\_thewitcher3.grid(row=7, column=1)  
 entry\_cloudgate = tk.Entry(window\_edit)  
 entry\_cloudgate.insert(0, df.loc[key][**'3DMark Cloud Gate'**])  
 entry\_cloudgate.grid(row=8, column=1)  
 entry\_firestrike = tk.Entry(window\_edit)  
 entry\_firestrike.insert(0, df.loc[key][**'3DMark Fire Strike'**])  
 entry\_firestrike.grid(row=9, column=1)  
 entry\_price = tk.Entry(window\_edit)  
 entry\_price.insert(0, df.loc[key][**'Средняя цена, ₽'**])  
 entry\_price.grid(row=10, column=1)  
 entry\_arch = tk.Entry(window\_edit)  
 entry\_arch.insert(0, df.loc[key][**'Архитектура'**])  
 entry\_arch.grid(row=11, column=1)  
 entry\_sli = tk.Entry(window\_edit)  
 entry\_sli.insert(0, df.loc[key][**'NVIDIA SLI'**])  
 entry\_sli.grid(row=12, column=1)  
 entry\_rtx = tk.Entry(window\_edit)  
 entry\_rtx.insert(0, df.loc[key][**'RTX'**])  
 entry\_rtx.grid(row=13, column=1)  
 entry\_freq = tk.Entry(window\_edit)  
 entry\_freq.insert(0, df.loc[key][**'Базовая тактовая частота, МГц'**])  
 entry\_freq.grid(row=14, column=1)  
 tk.Button(window\_edit, text=**'Ok'**, command=edit\_entry).grid(row=15, columnspan=2)  
 **else**:  
 **if '{' in** entry:  
 key = entry.split(**'}'**)[0][1:]  
 **else**:  
 key = entry.split(**' '**)[0]  
 **for** index **in** df.index:  
 **if** df.loc[index][**'Название'**] == key:  
 name = df.loc[index][**'Название'**]  
 arch = df.loc[index][**'Архитектура'**]  
 sli = df.loc[index][**'NVIDIA SLI'**]  
 rtx = df.loc[index][**'RTX'**]  
 freq = df.loc[index][**'Базовая тактовая частота, МГц'**]  
 **if** selected\_database.get() == 2:  
 tk.Label(window\_edit, text=**'Имя:'**).grid(row=1, column=0)  
 tk.Label(window\_edit, text=**'Архитектура'**).grid(row=2, column=0)  
 entry\_name = tk.Entry(window\_edit)  
 entry\_name.insert(0, name)  
 entry\_name.grid(row=1, column=1)  
 entry\_arch = tk.Entry(window\_edit)  
 entry\_arch.insert(0, arch)  
 entry\_arch.grid(row=2, column=1)  
 tk.Button(window\_edit, text=**'Ok'**, command=edit\_entry).grid(row=3, columnspan=2)  
 **elif** selected\_database.get() == 3:  
 tk.Label(window\_edit, text=**'Имя:'**).grid(row=1, column=0)  
 tk.Label(window\_edit, text=**'NVIDIA SLI'**).grid(row=2, column=0)  
 entry\_name = tk.Entry(window\_edit)  
 entry\_name.insert(0, name)  
 entry\_name.grid(row=1, column=1)  
 entry\_sli = tk.Entry(window\_edit)  
 entry\_sli.insert(0, sli)  
 entry\_sli.grid(row=2, column=1)  
 tk.Button(window\_edit, text=**'Ok'**, command=edit\_entry).grid(row=3, columnspan=2)  
 **elif** selected\_database.get() == 4:  
 tk.Label(window\_edit, text=**'Имя:'**).grid(row=1, column=0)  
 tk.Label(window\_edit, text=**'RTX'**).grid(row=2, column=0)  
 entry\_name = tk.Entry(window\_edit)  
 entry\_name.insert(0, name)  
 entry\_name.grid(row=1, column=1)  
 entry\_rtx = tk.Entry(window\_edit)  
 entry\_rtx.insert(0, rtx)  
 entry\_rtx.grid(row=2, column=1)  
 tk.Button(window\_edit, text=**'Ok'**, command=edit\_entry).grid(row=3, columnspan=2)  
 **elif** selected\_database.get() == 5:  
 tk.Label(window\_edit, text=**'Имя:'**).grid(row=1, column=0)  
 tk.Label(window\_edit, text=**'Базовая тактовая частота, МГц'**).grid(row=2, column=0)  
 entry\_name = tk.Entry(window\_edit)  
 entry\_name.insert(0, name)  
 entry\_name.grid(row=1, column=1)  
 entry\_freq = tk.Entry(window\_edit)  
 entry\_freq.insert(0, freq)  
 entry\_freq.grid(row=2, column=1)  
 tk.Button(window\_edit, text=**'Ok'**, command=edit\_entry).grid(row=3, columnspan=2)  
 **else**:  
 showinfo(ERROR\_MESSAGE, **"Не выбрана никакая запись"**)  
  
  
**def** open\_new\_window():  
 *"""  
 Функция для открытия окна добавления новой записи  
 Автор: Марков Д.Э.  
 """* **def** new\_entry():  
 *"""  
 Функция для добавления новой записи в базу данных  
 Автор: Марков Д.Э.  
 """* keys = [**'Название'**, **'Дата выхода'**, **'Конфигурация памяти, ГБ'**, **'Энергопотребление, Вт'**, **'Far Cry 5, FPS'**,  
 **'Fallout 4, FPS'**, **'The Witcher 3, FPS'**, **'3DMark Cloud Gate'**, **'3DMark Fire Strike'**,  
 **'Средняя цена, ₽'**, **'Архитектура'**, **'NVIDIA SLI'**, **'RTX'**, **'Базовая тактовая частота, МГц'**]  
 values = [  
 entry\_name.get(),  
 entry\_date.get(),  
 entry\_memory.get(),  
 entry\_power.get(),  
 entry\_farcry5.get(),  
 entry\_fallout4.get(),  
 entry\_thewitcher3.get(),  
 entry\_cloudgate.get(),  
 entry\_firestrike.get(),  
 entry\_price.get(),  
 entry\_arch.get(),  
 entry\_sli.get(),  
 entry\_rtx.get(),  
 entry\_freq.get()  
 ]  
 **if "" in** values:  
 showinfo(ERROR\_MESSAGE, **"Заполнены не все поля"**)  
 **else**:  
 entry = dict(zip([x **for** x **in** keys], [y **for** y **in** values]))  
 df.loc[(entry[**'Название'**], int(entry[**'Конфигурация памяти, ГБ'**])), list(df.columns)] = [entry[x] **for** x **in** df.columns]  
 update\_table()  
 window\_new.destroy()  
  
 window\_new = tk.Toplevel(window)  
 window\_new.resizable(0, 0)  
 tk.Label(window\_new, text=**'Новая запись в базе данных'**).grid(row=0, columnspan=2)  
 tk.Label(window\_new, text=**'Имя:'**).grid(row=1, column=0)  
 tk.Label(window\_new, text=**'Дата выхода:'**).grid(row=2, column=0)  
 tk.Label(window\_new, text=**'Конфигурация памяти, ГБ:'**).grid(row=3, column=0)  
 tk.Label(window\_new, text=**'Энергопотребление:'**).grid(row=4, column=0)  
 tk.Label(window\_new, text=**'Far Cry 5, FPS:'**).grid(row=5, column=0)  
 tk.Label(window\_new, text=**'Fallout 4, FPS:'**).grid(row=6, column=0)  
 tk.Label(window\_new, text=**'The Witcher 3, FPS:'**).grid(row=7, column=0)  
 tk.Label(window\_new, text=**'3DMark Cloud Gate:'**).grid(row=8, column=0)  
 tk.Label(window\_new, text=**'3DMark Fire Strike:'**).grid(row=9, column=0)  
 tk.Label(window\_new, text=**'Средняя цена, руб.:'**).grid(row=10, column=0)  
 tk.Label(window\_new, text=**'Архитектура:'**).grid(row=11, column=0)  
 tk.Label(window\_new, text=**'NVIDIA SLI:'**).grid(row=12, column=0)  
 tk.Label(window\_new, text=**'RTX:'**).grid(row=13, column=0)  
 tk.Label(window\_new, text=**'Базовая тактовая частота:'**).grid(row=14, column=0)  
  
 entry\_name = tk.Entry(window\_new)  
 entry\_name.grid(row=1, column=1)  
 entry\_date = tk.Entry(window\_new)  
 entry\_date.grid(row=2, column=1)  
 entry\_memory = tk.Entry(window\_new)  
 entry\_memory.grid(row=3, column=1)  
 entry\_power = tk.Entry(window\_new)  
 entry\_power.grid(row=4, column=1)  
 entry\_farcry5 = tk.Entry(window\_new)  
 entry\_farcry5.grid(row=5, column=1)  
 entry\_fallout4 = tk.Entry(window\_new)  
 entry\_fallout4.grid(row=6, column=1)  
 entry\_thewitcher3 = tk.Entry(window\_new)  
 entry\_thewitcher3.grid(row=7, column=1)  
 entry\_cloudgate = tk.Entry(window\_new)  
 entry\_cloudgate.grid(row=8, column=1)  
 entry\_firestrike = tk.Entry(window\_new)  
 entry\_firestrike.grid(row=9, column=1)  
 entry\_price = tk.Entry(window\_new)  
 entry\_price.grid(row=10, column=1)  
 entry\_arch = tk.Entry(window\_new)  
 entry\_arch.grid(row=11, column=1)  
 entry\_sli = tk.Entry(window\_new)  
 entry\_sli.grid(row=12, column=1)  
 entry\_rtx = tk.Entry(window\_new)  
 entry\_rtx.grid(row=13, column=1)  
 entry\_freq = tk.Entry(window\_new)  
 entry\_freq.grid(row=14, column=1)  
 tk.Button(window\_new, text=**'Добавить'**, command=new\_entry).grid(row=15, columnspan=2)  
  
  
**def** make\_list(df):  
 *"""  
 Функция для создания массива записей базы данных  
 Параметры: df - DataFrame  
 Возвращает кортеж из массива названий столбцов и массива записей  
 Автор: Хусаенов Т.И.  
 """* **if** selected\_database.get() == 2:  
 col = (**'Название'**, **'Архитектура'**)  
 s = list(set([(x, y) **for** x, y **in** zip(df[**'Название'**], df[**'Архитектура'**])]))  
 **elif** selected\_database.get() == 3:  
 col = (**'Название'**, **'NVIDIA SLI'**)  
 s = list(set([(x, y) **for** x, y **in** zip(df[**'Название'**], df[**'NVIDIA SLI'**])]))  
 **elif** selected\_database.get() == 4:  
 col = (**'Название'**, **'RTX'**)  
 s = list(set([(x, y) **for** x, y **in** zip(df[**'Название'**], df[**'RTX'**])]))  
 **elif** selected\_database.get() == 5:  
 col = (**'Название'**, **'Базовая тактовая частота, МГц'**)  
 s = list(set([(x, y) **for** x, y **in** zip(df[**'Название'**], df[**'Базовая тактовая частота, МГц'**])]))  
 **else**:  
 col = [  
 **'Название'**, **'Дата выхода'**, **'Конфигурация памяти, ГБ'**, **'Энергопотребление, Вт'**, **'Far Cry 5, FPS'**,  
 **'Fallout 4, FPS'**,  
 **'The Witcher 3, FPS'**, **'3DMark Cloud Gate'**, **'3DMark Fire Strike'**, **'Средняя цена, ₽'**]  
 s = list(  
 [[x1, x2, x3, x4, x5, x6, x7, x8, x9, x10] **for** x1, x2, x3, x4, x5, x6, x7, x8, x9, x10 **in** zip(df[col[0]],  
 df[col[1]],  
 df[col[2]],  
 df[col[3]],  
 df[col[4]],  
 df[col[5]],  
 df[col[6]],  
 df[col[7]],  
 df[col[8]],  
 df[col[9]])])  
 **return** col, s  
  
  
**def** update\_table(\*args):  
 *"""  
 Функция для обновления таблицы  
 Автор: Милосердов А.В.  
 """* **global** tree  
 tree.destroy()  
 arg = make\_list(df)  
 tree = new\_tree(arg[0], arg[1])  
  
 vsb = ttk.Scrollbar(orient=**"vertical"**,  
 command=tree.yview)  
 hsb = ttk.Scrollbar(orient=**"horizontal"**,  
 command=tree.xview)  
 tree.configure(yscrollcommand=vsb.set,  
 xscrollcommand=hsb.set)  
 tree.tag\_configure(**'mytag'**, background=**'#bef574'**)  
 vsb.place(x=1450, y=58, height=324)  
 tree.place(x=728, y=220, anchor=**'center'**)  
  
  
**def** new\_tree(col, l):  
 *"""  
 Функция для создания новой таблицы  
 Параметры: col - массив названий столбцов, l - массив записей  
 Возвращает новую таблицу  
 Автор: Хусаенов Т.И.  
 """* tree = ttk.Treeview(window, columns=col, show=**'headings'**, height=15, style=**"mystyle.Treeview"**)  
 **for** x **in** col:  
 tree.column(x, width=int(tkfont.Font().measure(x) / 1.1))  
 tree.heading(x, text=x, command=**lambda** c=x: sortby(tree, c, 0))  
 **for** x **in** l:  
 tree.insert(**""**, **"end"**, x, values=x, tags=(**'mytag'**,))  
 **return** tree  
  
  
**def** delete\_entries():  
 *"""  
 Функция для удаления записей из базы данных  
 Автор: Милосердов А.В.  
 """* **global** df  
 entries = tree.selection()  
 **if** len(entries) != 0:  
 **if** selected\_database.get() == 1:  
 **for** x **in** entries:  
 **if '{' in** x:  
 y = x.split(**'}'**)  
 k1 = y[0][1:]  
 x2 = y[1].split(**' '**)  
 k2 = int(float(x2[2]))  
 **else**:  
 y = x.split(**' '**)  
 k1 = y[0]  
 k2 = y[2]  
 df = df.drop((k1, int(k2)))  
 **else**:  
 **for** x **in** entries:  
 **if '{' in** x:  
 key = x.split(**'}'**)[0][1:]  
  
 **else**:  
 key = x.split(**' '**)[0]  
 **for** index **in** df.index:  
 **if** df.loc[index][**'Название'**] == key:  
 df = df.drop(index)  
 update\_table()  
 **else**:  
 showinfo(ERROR\_MESSAGE, **"Не выбрана ни одна запись"**)  
  
  
**def** save\_database():  
 *"""  
 Функция для сохранения базы данных  
 Автор: Хусаенов Т.И.  
 """* databin.write\_to\_binary(df, DATA\_SAVE\_ADDRESS)  
 showinfo(**"Успешно!"**, **"База данных сохранена"**)  
  
  
**def** restore\_database():  
 *"""  
 Функция для восстановления базы данных  
 Автор: Милосердов А.В.  
 """* **global** df  
 df = pd.read\_csv(READ\_ADDRESS)  
 df.index = ([(x, y) **for** x, y **in** zip(df[**'Название'**], df[**'Конфигурация памяти, ГБ'**])])  
 df.index = pd.MultiIndex.from\_tuples(df.index)  
 update\_table()  
 showinfo(**"Успешно!"**, **"База данных восстановлена"**)  
  
  
**try**:  
 df = databin.read\_from\_binary(DATA\_SAVE\_ADDRESS)  
**except**:  
 df = pd.read\_csv(READ\_ADDRESS)  
 df.index = ([(x, y) **for** x, y **in** zip(df[**'Название'**], df[**'Конфигурация памяти, ГБ'**])])  
 df.index = pd.MultiIndex.from\_tuples(df.index)  
OPTIONS = [0, 1, 2, 3, 4, 5]  
  
window = tk.Tk()  
window.geometry(MAIN\_WINDOW\_GEOMETRY)  
window.resizable(0, 0)  
window.title(MAIN\_WINDOW\_TITLE)  
window.iconbitmap(MAIN\_WINDOW\_ICO)  
style = ttk.Style()  
style.configure(**"mystyle.Treeview"**, highlightthickness=0, bd=0, font=(**'Calibri'**, 11)) *# Modify the font of the body*style.configure(**"mystyle.Treeview.Heading"**, font=(**'Calibri'**, 12, **'bold'**),  
 background=**'#76B900'**) *# Modify the font of the headings*style.layout(**"mystyle.Treeview"**, [(**'mystyle.Treeview.treearea'**, {**'sticky'**: **'nswe'**})]) *# Remove the borders*canvas = tk.Canvas(window, height=100, width=100)  
background\_image = tk.PhotoImage(file=BACKGROUND\_IMAGE)  
background\_label = tk.Label(window, image=background\_image)  
background\_label.image = background\_image  
background\_label.place(x=0, y=0, relwidth=1, relheight=1)  
canvas.place()  
  
btn\_new = tk.Button(window, width=BUTTON\_WIDTH, text=**'Добавить'**, command=open\_new\_window,  
 activebackground=FOREGROUNG\_COLOUR,  
 activeforeground=BUTTON\_ACTIVE\_FOREGROUND,  
 bg=BACKGROUND\_COLOUR, fg=FOREGROUNG\_COLOUR, font=TEXT\_OPTIONS)  
btn\_new.place(x=10, y=10)  
btn\_delete = tk.Button(window, width=BUTTON\_WIDTH, text=**'Удалить'**, command=delete\_entries,  
 activebackground=FOREGROUNG\_COLOUR,  
 activeforeground=BUTTON\_ACTIVE\_FOREGROUND,  
 bg=BACKGROUND\_COLOUR, fg=FOREGROUNG\_COLOUR, font=TEXT\_OPTIONS)  
btn\_delete.place(x=160, y=10)  
btn\_edit = tk.Button(window, width=BUTTON\_WIDTH, text=**'Редактировать'**, command=open\_edit\_window,  
 activebackground=FOREGROUNG\_COLOUR,  
 activeforeground=BUTTON\_ACTIVE\_FOREGROUND,  
 bg=BACKGROUND\_COLOUR, fg=FOREGROUNG\_COLOUR, font=TEXT\_OPTIONS)  
btn\_edit.place(x=310, y=10)  
  
btn\_save = tk.Button(window, width=BUTTON\_WIDTH, text=**'Сохранить'**, command=save\_database,  
 activebackground=FOREGROUNG\_COLOUR,  
 activeforeground=BUTTON\_ACTIVE\_FOREGROUND,  
 bg=BACKGROUND\_COLOUR, fg=FOREGROUNG\_COLOUR, font=TEXT\_OPTIONS)  
btn\_save.place(x=460, y=10)  
  
btn\_restore = tk.Button(window, width=BUTTON\_WIDTH, text=**'Восстановить'**, command=restore\_database,  
 activebackground=FOREGROUNG\_COLOUR,  
 activeforeground=BUTTON\_ACTIVE\_FOREGROUND,  
 bg=BACKGROUND\_COLOUR, fg=FOREGROUNG\_COLOUR, font=TEXT\_OPTIONS)  
btn\_restore.place(x=610, y=10)  
  
btn\_report = tk.Button(window, width=BUTTON\_WIDTH, text=**'Отчет'**, command=**lambda**: report(df),  
 activebackground=FOREGROUNG\_COLOUR,  
 activeforeground=BUTTON\_ACTIVE\_FOREGROUND,  
 bg=BACKGROUND\_COLOUR, fg=FOREGROUNG\_COLOUR, font=TEXT\_OPTIONS)  
btn\_report.place(x=760, y=10)  
  
scrollbar\_style = ttk.Style()  
scrollbar\_style.configure(**"My.Horizontal.TScrollbar"**, troughcolor=**"red"**)  
  
tree = ttk.Treeview(window, show=**'headings'**)  
  
selected\_database = IntVar(window)  
selected\_database.set(OPTIONS[1])  
options\_menu = ttk.OptionMenu(window, selected\_database, \*OPTIONS)  
options\_menu.place(x=910, y=15)  
selected\_database.trace(**'w'**, update\_table)  
update\_table()  
  
window.mainloop()

params.py

LIBRARY\_PATH = **'../Library/'**MAIN\_WINDOW\_GEOMETRY = **'1478x395'**MAIN\_WINDOW\_TITLE = **'База данных видеокарт NVIDIA'**MAIN\_WINDOW\_ICO = **'../Graphics/icon.ico'**BACKGROUND\_IMAGE = **'../Graphics/background.png'**BUTTON\_WIDTH = 13  
BUTTON\_ACTIVE\_FOREGROUND = **'#1A1918'**FOREGROUNG\_COLOUR = **'#76b900'**BACKGROUND\_COLOUR = **'#1A1918'**TEXT\_OPTIONS = (**'Roboto'**, 12, **'bold'**)  
READ\_ADDRESS = **'../Data/bd.csv'**DATA\_SAVE\_ADDRESS = **'../Data/data'**BAR\_PLOT\_ADDRESS = **'../Output/Столбчатая диаграмма.png'**HIST\_PLOT\_ADDRESS = **'../Output/Гистограмма.png'**BOX\_PLOT\_ADDRESS = **'../Output/Диаграмма Бокса-Вискерса.png'**SCATTER\_PLOT\_ADDRESS = **'../Output/Диаграмма рассеивания.png'**TEXT\_REPORT\_ADDRESS = **'../Output/output.txt'**ERROR\_MESSAGE = **'Ошибка'**

reportmodule.py

**from** tkinter.messagebox **import** showinfo  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
**from** params **import** \*  
  
power = **'Энергопотребление, Вт'**arch = **'Архитектура'  
  
  
def** bar\_plot(df):  
 *"""  
 Функция для создания и сохранения столбчатой диаграммы  
 Входные параметры: df - DataFrame, по которому строится диаграмма  
 Автор: Фурса Д.Д.  
 """* plt.figure(figsize=(20, 5))  
 plt.bar(df.loc[:, **'Название'**], df.loc[:, power], align=**'center'**, label=power)  
 plt.legend()  
 plt.ylabel(power)  
 plt.savefig(BAR\_PLOT\_ADDRESS)  
  
  
**def** hist\_plot(df):  
 *"""  
 Функция для создания и сохранения гистограммы  
 Входные параметры: df - DataFrame, по которому строится гистограмма  
 Автор: Фурса Д.Д.  
 """* plt.figure(figsize=(20, 5))  
 d = df.loc[:, power]  
 d.hist(bins=50)  
 plt.ylabel(**'Количетво'**)  
 plt.xlabel(power)  
 plt.savefig(HIST\_PLOT\_ADDRESS)  
  
  
**def** box\_plot(df):  
 *"""  
 Функция для создания и сохранения диаграммы Бокса-Вискера  
 Входные параметры: df - DataFrame, по которому строится диаграмма  
 Автор: Фурса Д.Д.  
 """* plt.figure(figsize=(20, 5))  
 d = df.loc[:, power]  
 plt.boxplot(d, notch=**True**, patch\_artist=**True**)  
 plt.ylabel(power)  
 plt.savefig(BOX\_PLOT\_ADDRESS)  
  
  
**def** scatter\_plot(df):  
 *"""  
 Функция для создания и сохранения диаграммы рассеивания  
 Входные параметры: df - DataFrame, по которому строится диаграмма  
 Автор: Фурса Д.Д.  
 """* area = np.pi \* 3  
 plt.figure(figsize=(10, 5))  
 plt.scatter(df.loc[:, **'Конфигурация памяти, ГБ'**], df.loc[:, **'Fallout 4, FPS'**], s=area)  
 plt.ylabel(**'Fallout 4, FPS'**)  
 plt.xlabel(**'Конфигурация памяти, ГБ'**)  
 plt.savefig(SCATTER\_PLOT\_ADDRESS)  
  
  
**def** report(df):  
 *"""  
 Функция для создания и сохранения текстового отчета и вызова функций создания графиков  
 Входные параметры: df - DataFrame, по которому создается отчет  
 Автор: Марков Д.Э.  
 """* entries\_count = len(df.index)  
 memory\_avg = round(np.average(df[**'Конфигурация памяти, ГБ'**]), 2)  
 memory\_disp = round(np.var(df[**'Конфигурация памяти, ГБ'**]), 2)  
 power\_avg = round(np.average(df[**'Энергопотребление, Вт'**]), 2)  
 power\_disp = round(np.var(df[**'Энергопотребление, Вт'**]), 2)  
 farcry5\_avg = round(np.average(df[**'Far Cry 5, FPS'**]), 2)  
 farcry5\_disp = round(np.var(df[**'Far Cry 5, FPS'**]), 2)  
 fallout4\_avg = round(np.average(df[**'Fallout 4, FPS'**]), 2)  
 fallout4\_disp = round(np.var(df[**'Fallout 4, FPS'**]), 2)  
 witcher3\_avg = round(np.average(df[**'The Witcher 3, FPS'**]), 2)  
 witcher3\_disp = round(np.var(df[**'The Witcher 3, FPS'**]), 2)  
 cloudgate\_avg = round(np.average(df[**'3DMark Cloud Gate'**]), 2)  
 cloudgate\_disp = round(np.var(df[**'3DMark Cloud Gate'**]), 2)  
 firestrike\_avg = round(np.average(df[**'3DMark Fire Strike'**]), 2)  
 firestrike\_disp = round(np.var(df[**'3DMark Fire Strike'**]), 2)  
 file = open(TEXT\_REPORT\_ADDRESS, **'w'**)  
 file.write(**'Количество записей: '** + str(entries\_count) + **'\n'**)  
 file.write(**'Среднее значение памяти: '** + str(memory\_avg) + **'\n'**)  
 file.write(**'Дисперсия памяти: '** + str(memory\_disp) + **'\n'**)  
 file.write(**'Среднее значение энергопотребления: '** + str(power\_avg) + **'\n'**)  
 file.write(**'Дисперсия энергопотребления: '** + str(power\_disp) + **'\n'**)  
 file.write(**'Среднее значение FPS в Far Cry 5: '** + str(farcry5\_avg) + **'\n'**)  
 file.write(**'Дисперсия FPS в Far Cry 5: '** + str(farcry5\_disp) + **'\n'**)  
 file.write(**'Среднее значение FPS в Fallout 4: '** + str(fallout4\_avg) + **'\n'**)  
 file.write(**'Дисперсия FPS в Fallout 4: '** + str(fallout4\_disp) + **'\n'**)  
 file.write(**'Среднее значение FPS в The Witcher 3: '** + str(witcher3\_avg) + **'\n'**)  
 file.write(**'Дисперсия FPS в The Witcher 3: '** + str(witcher3\_disp) + **'\n'**)  
 file.write(**'Среднее значение 3DMark Cloud Gate: '** + str(cloudgate\_avg) + **'\n'**)  
 file.write(**'Дисперсия 3DMark Cloud Gate: '** + str(cloudgate\_disp) + **'\n'**)  
 file.write(**'Среднее значение 3DMark Fire Strike: '** + str(firestrike\_avg) + **'\n'**)  
 file.write(**'Дисперсия 3DMark Fire Strike: '** + str(firestrike\_disp) + **'\n'**)  
 file.close()  
 bar\_plot(df)  
 hist\_plot(df)  
 box\_plot(df)  
 scatter\_plot(df)  
 showinfo(**"Успешно!"**, **"Отчёт сохранён в папке Output!"**)