**Руководство разработчика** **к специализированному информационно-аналитическому приложению “Процессоры”**

Разработчики:

Балескин В.

Колесов Д.

Литвиненко А.

Тарунтаева А.

**Оглавление**

[**1.** **Описание библиотек, используемых в проекте** 3](#_Toc10907070)

[**2.** **Описание структуры каталогов** 3](#_Toc10907071)

[**3.** **Структура приложения** 4](#_Toc10907072)

[**4.** **Листинг** 4](#_Toc10907073)

[**5.** **Технические требования** 5](#_Toc10907074)

# **Описание библиотек, используемых в проекте**

Приложение создано с помощью языка Python 3.7, и в этом приложении используются следующие библиотеки:

* Numpy (ver. 1.16.4) - это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами.
* Pandas (ver. 0.24.3) - это Python библиотека для анализа и обработки данных. Она позволяет легко исследовать данные.
* Matplotlib(ver. 3.1.0) - это библиотека двумерной графики для языка программирования Python, с помощью которой можно создавать высококачественные рисунки различных форматов.
* Tkinter - это графическая библиотека на основе Tk, которая входит в стандартную библиотеку Python. Tk является библиотекой базовых элементов графического интерфейса для языка Tcl. Tkinter реализован как оболочка Python для Tcl интерпретатора, встроенного в интерпретатор Python.

# **Описание структуры каталогов**

Все составные части программы хранятся в каталоге «work». Внутри этого каталога существует 6 подкаталогов:

* Data – здесь хранятся входные данные для приложения: базы данных: cores.csv, processors\_v.csv, processors\_q.csv,
* Graphics – каталог, в котором хранятся графические отчеты, полученные в ходе работы приложения
* Library – каталог для библиотек, используемых в этом приложении
* Notes – папка, в которой находятся руководства пользователя и разработчика
* Output – каталог, в котором хранятся текстовые отчеты, полученные в ходе работы приложения
* Scripts – каталог, внутри которого лежат файл-исполнитель и файл настроек config.ini

# **Структура приложения**

Приложение состит из двух частей:

1. Агрегатор данных справочников и сессионной БД, он включает в себя:
   1. Database.py – представляет БД в системе, замещает функционал DataFrame, там где это нужно
   2. DatabaseParser – хранит множество основных справочников и сессионный справочник. Обрабатывает запросы к ним и позволяет получить сессионный справочник
   3. ConfigurationParser и связанный с ним класс Block - позволяют считывать и представлять файл конфигурации в системе
2. Графический интерфейс, он включает в себя:

2.1) main.py – содержит в себе код для реализации графического интерфейса, состоящего из основного окна с раположенным в нем отображением БД и функциональными кнопками, а также всплывающего окна с функциями построения графических и текстовых отчетов

# **Листинг**

Файл main.py находится в каталоге Scripts

# -\*- coding: utf-8 -\*-

"""

Created on Wed Apr 3 23:37:20 2019

Автором всех классов и методов является Колесов, если иное не обозначено в докстринге этого класса или метода

"""

import sys

sys.path.insert(0, '..\\..\\')

sys.path.insert(1, '..\\')

sys.path.insert(2, '..\\Library\\')

from tkinter import messagebox as mb

import tkinter as tk

import pandas as pd

from tkinter import ttk

from tkinter import Canvas

from tkinter import BooleanVar

from tkinter import StringVar

import matplotlib.pyplot as plt

import numpy as np

from collections import Counter

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from Work.Library.DataBase import DataBase

from Work.Library.database\_interaction.DatabaseParser import DatabaseParser

from Work.Library.configuration\_parser import \*

class Main(tk.Frame):

"""Основное окно"""

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

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

self.init\_main(atributes)

self.dp = dp

def init\_main(self, atributes):

toolbar = tk.Frame(bg='#d7d8e0', bd=2)

toolbar.pack(side=tk.TOP, fill=tk.X)

mainmenu = tk.Menu(root)

root.config(menu=mainmenu)

addmenu = tk.Menu(mainmenu, tearoff=0)

addmenu.add\_command(label='Добавить объект', command=self.open\_add\_object)

addmenu.add\_command(label='Добавить атрибут', command=self.open\_add\_atribute)

deletemenu = tk.Menu(mainmenu, tearoff=0)

deletemenu.add\_command(label='Удалить объект', command=self.delete\_object)

deletemenu.add\_command(label='Удалить атрибут', command=self.open\_delete\_dialog)

edit = tk.Menu(mainmenu, tearoff=0)

editmenu = tk.Menu(edit, tearoff=0)

editmenu.add\_command(label='Редактировать объект', command=self.open\_edit\_object)

editmenu.add\_command(label='Редактировать атрибут', command=self.open\_edit\_atribute)

showmenu = tk.Menu(mainmenu, tearoff=0)

showmenu.add\_command(label='Новая сессия', command=self.open\_show\_db)

showmenu.add\_command(label='Загрузить сессию', command=self.open\_download\_session)

edit.add\_cascade(label="Добавить", menu=addmenu)

edit.add\_cascade(label="Удалить", menu=deletemenu)

edit.add\_cascade(label="Редактировать", menu=editmenu)

mainmenu.add\_cascade(label="Действие", menu=edit)

mainmenu.add\_cascade(label="База данных", menu=showmenu)

mainmenu.add\_command(label='Сохранить', command=self.open\_save\_db)

mainmenu.add\_command(label='Справка', command=self.open\_info)

mainmenu.add\_command(label='Отчеты', command=self.open\_reports)

mainmenu.add\_command(label='×', command=self.destroy\_links)

self.scrollbar1 = tk.Scrollbar(self, orient=tk.HORIZONTAL)

self.scrollbar1.pack(side='bottom', fill='x')

self.scrollbar2 = tk.Scrollbar(self, orient=tk.VERTICAL)

self.scrollbar2.pack(side='right', fill='y')

self.tree = ttk.Treeview(self, columns=[i for i in atributes], height=100,

xscrollcommand=self.scrollbar1.set,

yscrollcommand=self.scrollbar2.set)

self.tree.pack(side='top')

for i in atributes:

self.tree.column(i, anchor=tk.CENTER)

for i in atributes:

self.tree.heading(i, text=i)

self.tree.heading("#0", text="Model")

self.scrollbar1.config(command=self.tree.xview)

self.scrollbar2.config(command=self.tree.yview)

self.update\_db()

def open\_reports(self):

self.app1 = Reports()

def download\_session(self, entry):

# print('11111')

# print(entry.get()+'!')

dp.parse("LOAD", entry.get())

self.update\_db()

def destroy\_links(self):

self.tree.focus\_set()

self.update\_db()

def update\_db(self):

self.tree.config(columns=[i for i in dp.working\_db.get\_db().columns])

for i in dp.working\_db.get\_db().columns:

self.tree.column(i, anchor=tk.CENTER)

self.tree.heading(i, text=i)

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

[self.tree.insert('', 'end', values=row\_data[1], text=row\_data[0])

for row\_data in zip(dp.working\_db.get\_db().index.tolist(),

dp.working\_db.get\_db().values.tolist())]

def add\_object(self, key\_entry, entries):

dp.parse("APPEND", [], [key\_entry.get()], [{i: j for i, j in

zip(dp.working\_db.get\_db().columns, [k.get() for k in entries])}],

None)

self.update\_db()

def add\_atribute(self, entry, combobox):

dp.parse("APPEND", [entry.get()], [], [], None)

self.update\_db()

def delete\_object(self):

if self.tree.selection() == ():

mb.showerror('Ошибка', 'Должны быть выбраны объекты')

else:

for i in self.tree.selection():

dp.parse("DELETE", [], [self.tree.item(i)['text']])

self.update\_db()

def delete\_atribute(self, entries):

dp.parse("DELETE", [dp.working\_db.get\_db().columns[i] for i in entries], [])

self.update\_db()

def edit\_object(self, entries):

for i, j in zip(dp.working\_db.get\_db().columns, entries):

dp.parse("CHANGE", i, self.tree.item(self.tree.selection()[0])['text'], str(j.get()))

self.tree.focus\_set()

self.update\_db()

def select\_all(self, variables):

for i in variables:

if i.get() is False:

[j.set(True) for j in variables]

else:

[j.set(False) for j in variables]

def edit\_atribute(self, combobox, entry):

dp.parse("RENAME", {combobox.get(): entry.get()}, "columns")

self.update\_db()

def view\_records(self, variables, atributes):

dp.parse("DISPLAY", ["-i"] + [i for i, j in zip(atributes, variables) if j.get()], [None, None], None)

self.update\_db()

def save\_db(self, entry):

dp.parse("STORE", entry.get())

def open\_download\_session(self):

DownloadSession()

def open\_info(self):

pass

def open\_add\_object(self):

AddObject(atributes)

def open\_add\_atribute(self):

AddAtribute()

def open\_delete\_dialog(self):

DeleteAtribute()

def open\_edit\_object(self):

if self.tree.selection() == ():

mb.showerror('Ошибка', 'Должен быть выбран объект')

else:

EditObject()

def open\_edit\_atribute(self):

EditAtribute()

def open\_show\_db(self):

ShowDB(atributes)

def open\_save\_db(self):

SaveDB()

class DownloadSession(tk.Toplevel):

"""Загрузка сессии из файла Авторы - Тарунтаева"""

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

"""Автор этого метода - Колесов"""

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

self.init\_child()

self.view = app

def init\_child(self):

self.title('Загрузка сессии')

self.geometry('300x150+400+300')

self.resizable(False, False)

self.label\_name1 = tk.Label(self, text='Введите название существующей базы даных:')

self.label\_name1.pack(side='top', pady=7)

self.label\_name2 = tk.Label(self, text='(без расширения)')

self.label\_name2.pack(side='top')

self.entry = ttk.Entry(self, width=30)

self.entry.pack(side='top', pady=7)

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

self.btn\_cancel.place(x=210, y=110)

self.btn\_ok = ttk.Button(self, text='Загрузить', command=self.destroy)

self.btn\_ok.place(x=130, y=110)

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

self.grab\_set()

self.focus\_set()

class DeleteAtribute(tk.Toplevel):

"""Удаление атрибута Автор - Тарунтаева"""

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

"""Автор этого метода - Колесов"""

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

self.init\_child()

self.view = app

def init\_child(self):

self.title('Удалить атрибуты')

self.geometry('200x300+400+300')

self.resizable(False, False)

myframe = tk.Frame(self, width=200, height=250)

myframe.pack(anchor='nw')

myscrollbar = tk.Scrollbar(myframe, orient="vertical")

myscrollbar.pack(side='right', fill='y')

lbox = tk.Listbox(myframe, width=30, height=16, selectmode=tk.EXTENDED, yscrollcommand=myscrollbar.set)

myscrollbar.configure(command=lbox.yview)

lbox.pack(anchor='nw')

[lbox.insert(tk.END, i) for i in dp.working\_db.get\_db().columns]

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

btn\_cancel.place(x=20, y=265)

btn\_ok = ttk.Button(self, text='Удалить', command=self.destroy)

btn\_ok.place(x=100, y=265)

btn\_ok.bind('<Button-1>', lambda event: self.view.delete\_atribute(lbox.curselection()))

self.grab\_set()

self.focus\_set()

class AddObject(tk.Toplevel):

"""Добавление объекта"""

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

"""Автор - Балескин - только этого метода"""

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

self.init\_child(atributes)

self.view = app

def init\_child(self, atributes):

self.title('Добавить новый объект')

self.geometry('350x300+400+300')

self.resizable(False, False)

self.myframe = tk.Frame(self, width=300, height=300, bd=0)

self.myframe.place(x=0, y=0)

canvas = Canvas(self.myframe, bd=0, highlightthickness=0)

self.frame = tk.Frame(canvas, bd=0)

myscrollbar = tk.Scrollbar(self.myframe, orient="vertical", command=canvas.yview)

canvas.configure(yscrollcommand=myscrollbar.set)

myscrollbar.pack(side="right", fill="y")

canvas.pack(side="left")

canvas.create\_window((0, 10), window=self.frame, anchor='nw')

self.frame.bind("<Configure>",

lambda event: canvas.configure(scrollregion=canvas.bbox("all"), width=330, height=250))

self.Key\_label = ttk.Label(self.frame, text='Model:')

self.Key\_label.grid(row=0, column=2, pady=3, padx=30, sticky='w')

self.Key\_entry = ttk.Entry(self.frame)

self.Key\_entry.grid(row=0, column=3, padx=30)

self.Labels = [ttk.Label(self.frame, text=i + ':') for i in dp.working\_db.get\_db().columns]

self.Entries = [ttk.Entry(self.frame) for i in range(len(dp.working\_db.get\_db().columns))]

for i, j in zip([i for i in range(len(dp.working\_db.get\_db().columns) + 1)[1:]], self.Labels):

j.grid(row=i, column=2, pady=3, padx=30, sticky='w')

for i, j in zip([i for i in range(len(dp.working\_db.get\_db().columns) + 1)[1:]], self.Entries):

j.grid(row=i, column=3, pady=3, padx=30)

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

btn\_cancel.place(x=240, y=265)

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

self.btn\_ok.place(x=160, y=265)

self.btn\_ok.bind('<Button-1>', lambda event: self.view.add\_object(self.Key\_entry, self.Entries))

self.grab\_set()

self.focus\_set()

class AddAtribute(tk.Toplevel):

"""Добавление атрибута Автор - Балескин"""

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

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

self.init\_child(atributes)

self.view = app

def init\_child(self, atributes):

self.title('Добавить новый атрибут')

self.geometry('300x200+400+300')

self.resizable(False, False)

self.label\_name = tk.Label(self, text='Имя атрибута:')

self.label\_name.place(x=30, y=40)

self.entry = ttk.Entry(self)

self.entry.place(x=130, y=40)

self.label\_type = tk.Label(self, text='Тип данных:')

self.label\_type.place(x=30, y=90)

self.combobox = ttk.Combobox(self, values=['Число', 'Строка'], width=17)

self.combobox.place(x=130, y=90)

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

self.btn\_cancel.place(x=210, y=160)

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

self.btn\_ok.place(x=130, y=160)

self.btn\_ok.bind('<Button-1>', lambda event: self.view.add\_atribute(self.entry, self.combobox))

self.grab\_set()

self.focus\_set()

class EditAtribute(tk.Toplevel):

"""Окно изменения атрибута"""

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

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

self.init\_child()

self.view = app

def init\_child(self):

self.title('Редактировать атрибут')

self.geometry('300x200+400+300')

self.resizable(False, False)

self.label\_name = tk.Label(self, text='Новое название:')

self.label\_name.place(x=30, y=90)

self.entry = ttk.Entry(self)

self.entry.place(x=130, y=90)

self.label\_type = tk.Label(self, text='Атрибут:')

self.label\_type.place(x=30, y=40)

self.combobox = ttk.Combobox(self, values=[i for i in dp.working\_db.get\_db().columns], width=17)

self.combobox.place(x=130, y=40)

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

self.btn\_cancel.place(x=210, y=160)

self.btn\_ok = ttk.Button(self, text='Сохранить', command=self.destroy)

self.btn\_ok.place(x=130, y=160)

self.btn\_ok.bind('<Button-1>', lambda event: self.view.edit\_atribute(self.combobox, self.entry))

self.grab\_set()

self.focus\_set()

class EditObject(tk.Toplevel):

"""Окно изменения объекта"""

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

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

self.view = app

self.init\_child(self.view)

def init\_child(self, view):

self.title('Редактировать объект')

self.geometry('350x300+400+300')

self.resizable(False, False)

self.myframe = tk.Frame(self, width=300, height=300, bd=0)

self.myframe.place(x=0, y=0)

canvas = Canvas(self.myframe, bd=0, highlightthickness=0)

self.frame = tk.Frame(canvas, bd=0)

myscrollbar = tk.Scrollbar(self.myframe, orient="vertical", command=canvas.yview)

canvas.configure(yscrollcommand=myscrollbar.set)

myscrollbar.pack(side="right", fill="y")

canvas.pack(side="left")

canvas.create\_window((0, 10), window=self.frame, anchor='nw')

self.frame.bind("<Configure>",

lambda event: canvas.configure(scrollregion=canvas.bbox("all"), width=330, height=250))

self.Variables = [StringVar() for i in range(len(dp.working\_db.get\_db().columns))]

self.Labels = [ttk.Label(self.frame, text=i + ':') for i in dp.working\_db.get\_db().columns]

self.Entries = [ttk.Entry(self.frame, textvariable=j) for i, j in

zip(range(len(dp.working\_db.get\_db().columns)), self.Variables)]

[self.Variables[j].set(i) for i, j in

zip(view.tree.item(view.tree.selection())['values'], range(len(self.Variables)))]

for i, j in zip([i for i in range(len(dp.working\_db.get\_db().columns))], self.Labels):

j.grid(row=i, column=2, pady=3, padx=30, sticky='w')

for i, j in zip([i for i in range(len(dp.working\_db.get\_db().columns))], self.Entries):

j.grid(row=i, column=3, pady=3, padx=30)

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

self.btn\_cancel.place(x=240, y=265)

self.btn\_ok = ttk.Button(self, text='Сохранить', command=self.destroy)

self.btn\_ok.place(x=160, y=265)

self.btn\_ok.bind('<Button-1>', lambda event: self.view.edit\_object(self.Entries))

self.btn\_cancel.bind('<Button-1>', lambda event: self.view.destroy\_links())

self.grab\_set()

self.focus\_set()

class ShowDB(tk.Toplevel):

"""Отображение и обновление БД"""

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

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

self.init\_child(atributes)

self.view = app

def init\_child(self, atributes):

self.title('Отобразить')

self.geometry('200x300+400+300')

self.resizable(False, False)

self.myframe = tk.Frame(self, width=200, height=300, bd=0)

self.myframe.place(x=0, y=0)

canvas = Canvas(self.myframe, bd=0, highlightthickness=0)

self.frame = tk.Frame(canvas, bd=0)

myscrollbar = tk.Scrollbar(self.myframe, orient="vertical", command=canvas.yview)

canvas.configure(yscrollcommand=myscrollbar.set)

myscrollbar.pack(side="right", fill="y")

canvas.pack(side="left")

canvas.create\_window((0, 10), window=self.frame, anchor='nw')

self.frame.bind("<Configure>",

lambda event: canvas.configure(scrollregion=canvas.bbox("all"), width=180, height=250))

self.Variables = [BooleanVar() for i in range(len(atributes))]

self.Checkbuttons = [tk.Checkbutton(self.frame, text=i, variable=j) for i, j in zip(atributes, self.Variables)]

self.checkbutton\_stringvar = BooleanVar()

self.checkbutton = tk.Checkbutton(self.frame, text='Выбрать все', variable=self.checkbutton\_stringvar)

self.checkbutton.grid(row=1, column=2, pady=3, padx=20, sticky='w')

for i, j in zip([i for i in range(len(self.Checkbuttons))], self.Checkbuttons):

j.grid(row=i + 1, column=2, pady=3, padx=20, sticky='w')

self.checkbutton.bind('<Button-1>', lambda event: self.view.select\_all(self.Variables))

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

btn\_cancel.place(x=20, y=265)

btn\_ok = ttk.Button(self, text='Обновить', command=self.destroy)

btn\_ok.place(x=100, y=265)

btn\_ok.bind('<Button-1>', lambda event: self.view.view\_records(self.Variables, atributes))

self.grab\_set()

self.focus\_set()

class SaveDB(tk.Toplevel):

"""Сохранение сессионной БД"""

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

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

self.init\_child()

self.view = app

def init\_child(self):

self.title('Сохранение базы данных')

self.geometry('300x150+400+300')

self.resizable(False, False)

self.label\_name1 = tk.Label(self, text='Введите название новой базы даных:')

self.label\_name1.pack(side='top', pady=7)

self.label\_name2 = tk.Label(self, text='(без расширения)')

self.label\_name2.pack(side='top')

self.entry = ttk.Entry(self, width=30)

self.entry.pack(side='top', pady=7)

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

self.btn\_cancel.place(x=210, y=110)

self.btn\_ok = ttk.Button(self, text='Сохранить', command=self.destroy)

self.btn\_ok.place(x=130, y=110)

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

self.grab\_set()

self.focus\_set()

class Reports(tk.Toplevel):

"""Автор: Литвиненко"""

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

"""Автор этого метода - Тарунтаева"""

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

self.view = app

self.state('zoomed')

self.dataframe = self.view.dp.working\_db.get\_db()

self.attributes = list(self.dataframe.keys())

self.v\_attrs = [str(attr) for attr in self.attributes if self.is\_val\_attr(str(attr))]

self.q\_attrs = [str(attr) for attr in self.attributes if attr not in self.v\_attrs]

self.figure = plt.Figure(figsize=(5, 4), dpi=75)

self.ax = self.figure.add\_subplot(111)

self.to\_delete = []

self.additional\_clear = []

self.combos = []

self.labels = []

self.to\_file = None

self.report\_type = "NONE"

self.text\_type = "NONE"

self.init\_child(self.view)

def is\_val\_attr(self, attr):

"""Проверяет, является ли атрибут количественным"""

val = str(self.dataframe[attr].iloc[0])

dots = 0

for ch in val:

if not (str.isdigit(ch) or (ch == '.')):

return False

if ch == '.':

dots += 1

if dots > 1:

return False

return True

def init\_child(self, view):

"""Формирует окно"""

self.title('Проект по питону')

self.geometry('1000x550')

self.plot\_area\_frame = tk.LabelFrame(self, text='Plot Area')

self.plot\_area\_frame.place(x=500, y=10, height=500, width=450)

self.settings\_area\_frame = tk.LabelFrame(self, text='Settings Area')

self.settings\_area\_frame.place(x=100, y=10, height=500, width=350)

self.menu = tk.Menu(self)

new\_item = tk.Menu(self.menu, tearoff=0)

new\_item.add\_command(label='Новый')

new\_item.add\_separator()

new\_item.add\_command(label='Изменить')

self.menu.add\_cascade(label='Файл', menu=new\_item)

self.btn\_save = ttk.Button(self.settings\_area\_frame, text='Сохранить', command=self.open\_save\_report)

self.btn\_save.pack(side='bottom', anchor='se')

value = StringVar()

self.combo = ttk.Combobox(self.settings\_area\_frame, textvariable=value)

lbl = tk.Label(self.settings\_area\_frame, text='Выберите вид отчета:')

lbl.place(x=10, y=7, width=150)

self.combo['values'] = ("Столбчатая диаграмма(кач-кач)", "Гистограмма(кол-кач)",

"Диаграмма Бокса-Вискера(кол-кач)", "Диаграмма рассеивания(2 кол - кач)",

"Сводная таблица (кач-кач)", "Набор осн. опис. стат")

self.combo.current(0)

self.combo.place(x=10, y=30, width=200)

self.choose\_btn = ttk.Button(self.settings\_area\_frame, text='Выбрать', command=self.click)

self.choose\_btn.place(x=250, y=30, width=70)

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

self.btn\_cancel.pack(anchor='se', side='bottom')

# self.btn\_cancel.bind('<Button-1>', lambda event: self.view.destroy\_links())

self.grab\_set()

self.focus\_set()

def paint\_figure(self):

"""Отрисовывает фигуру"""

self.report\_type = "GRAPH"

canvas = FigureCanvasTkAgg(self.figure, master=self.plot\_area\_frame)

canvas.draw()

canvas.get\_tk\_widget().place(x=20, y=10, height=450, width=400)

self.to\_file = self.figure

def clear(self):

"""Разрушает фигуру"""

self.figure.clear()

self.figure = plt.Figure(figsize=(5, 4), dpi=75)

self.ax = self.figure.add\_subplot(111)

def add\_combo(self, attrs, text):

"""Добавляет комбобокс"""

ys = [73, 116, 159, 202]

self.combos.append(ttk.Combobox(self.settings\_area\_frame))

self.labels.append(tk.Label(self.settings\_area\_frame, text=text, anchor='w'))

self.to\_delete.append(self.combos[-1])

self.combos[-1]["values"] = attrs

self.combos[-1].current(0)

self.combos[-1].place(x=10, y=ys[len(self.combos) - 1] + 2, height=20, width=200)

self.labels[-1].place(x=10, y=ys[len(self.combos) - 1] - 21, height=20)

self.to\_delete.append(self.labels[-1])

def click(self):

"""Обрабатывает нажатие кнопки выбора типа отчета Автор этого метода - Тарунтаева"""

for i in self.to\_delete:

i.destroy()

for i in self.additional\_clear:

i.destroy()

self.to\_delete.clear()

self.additional\_clear.clear()

self.combos.clear()

self.clear()

btn2 = ttk.Button(self.settings\_area\_frame, text='Построить')

if self.combo.get() == "Диаграмма рассеивания(2 кол - кач)":

self.add\_combo(self.v\_attrs, text="Выберите количественный атрибут")

self.add\_combo(self.v\_attrs, text="Выберите количественный атрибут")

self.add\_combo(self.q\_attrs, text="Выберите качественный атрибут")

btn2.bind("<Button-1>", self.build\_scatter)

if self.combo.get() == "Столбчатая диаграмма(кач-кач)":

self.add\_combo(self.q\_attrs, text="Выберите качественный атрибут")

self.add\_combo(self.q\_attrs, text="Выберите качественный атрибут")

btn2.bind("<Button-1>", self.build\_bar)

if self.combo.get() == "Гистограмма(кол-кач)":

self.add\_combo(self.v\_attrs, text="Выберите количественный атрибут")

self.add\_combo(self.q\_attrs, text="Выберите качественный атрибут")

btn2.bind("<Button-1>", self.build\_hist)

if self.combo.get() == "Диаграмма Бокса-Вискера(кол-кач)":

self.add\_combo(self.v\_attrs, text="Выберите количественный атрибут")

self.add\_combo(self.q\_attrs, text="Выберите качественный атрибут")

btn2.bind("<Button-1>", self.build\_box)

if self.combo.get() == "Сводная таблица (кач-кач)":

self.add\_combo(self.q\_attrs, text="Выберите качественный атрибут")

self.add\_combo(self.q\_attrs, text="Выберите качественный атрибут")

self.add\_combo(self.v\_attrs, text="Выберите количественный атрибут для аггрегации")

self.add\_combo(["mean", "sum", "standard deviation"], text="Выберите метод аггрегации")

btn2.bind("<Button-1>", self.build\_pivot)

if self.combo.get() == "Набор осн. опис. стат":

self.listbox = tk.Listbox(self.settings\_area\_frame, selectmode='extended', height=5)

[self.listbox.insert('end', i) for i in self.v\_attrs]

self.label = tk.Label(self.settings\_area\_frame, text='Выберите атрибуты:')

self.label.place(x=10, y=70)

self.to\_delete.append(self.label)

self.listbox.place(x=10, y=95)

self.to\_delete.append(self.listbox)

btn2.bind("<Button-1>", lambda event: self.major\_desc\_stats(self.listbox.curselection()))

btn2.place(x=250, y=73, width=70)

def build\_bar(self, event=None):

"""Построение столбчатой диаграммы"""

first\_attr = self.combos[0].get()

second\_attr = self.combos[1].get()

vals\_prep = {i: set() for i in set(self.dataframe[first\_attr].values)}

for i in self.dataframe[[first\_attr, second\_attr]].values:

vals\_prep[i[0]].add(i[1])

self.clear()

vals = {i: len(vals\_prep[i]) for i in vals\_prep.keys()}

indices = list(set(self.dataframe[first\_attr].values))

d = pd.DataFrame({first\_attr: indices, second\_attr: list(vals.values())})

d.plot.bar(ax=self.ax, x=first\_attr, y=second\_attr)

# add label, saying "Измеряется количество разных значений второго атрибута"

self.lblLegend = tk.Label(self, text="Измеряется количество\nразных значений второго атрибута")

self.lblLegend.place(x=1000, y=10, height=60, width=200)

self.to\_delete.append(self.lblLegend)

self.paint\_figure()

def build\_hist(self, event=None):

"""Построение гистограммы"""

first\_attr = self.combos[0].get()

second\_attr = self.combos[1].get()

self.clear()

self.dataframe.hist(ax=self.ax, column=first\_attr, by=second\_attr,

bins=int(1 + np.log2(len(self.dataframe[first\_attr].values))))

self.paint\_figure()

def build\_box(self, event=None):

"""Построение диаграммы бокса-вискера"""

first\_attr = self.combos[0].get()

second\_attr = self.combos[1].get()

self.clear()

self.dataframe.boxplot(ax=self.ax, by=second\_attr, column=first\_attr)

self.paint\_figure()

def build\_scatter(self, event=None):

"""построение диаграммы рассеивания"""

first\_attr = self.combos[0].get()

second\_attr = self.combos[1].get()

third\_attr = self.combos[2].get()

s = list(set(self.dataframe[third\_attr].values))

colormap = {s[i]: i for i in range(len(s))}

colors = [colormap[element] for element in self.dataframe[third\_attr].values]

ind = np.arange(5)

width = .5

self.clear()

self.dataframe.plot.scatter(ax=self.ax, x=first\_attr, y=second\_attr, c=colors, colormap='viridis')

text = ""

for item in colormap.items():

text += str(item[0]) + ":" + str(item[1]) + "\n"

self.lblLegend = tk.Label(self, text=text)

self.lblLegend.place(x=1000, y=10, height=len(colormap) \* 20, width=200)

self.to\_delete.append(self.lblLegend)

self.paint\_figure()

def build\_pivot(self, event=None):

"""Построение сводной таблицы"""

for i in self.additional\_clear:

i.destroy()

self.additional\_clear.clear()

first\_attr = self.combos[0].get()

second\_attr = self.combos[1].get()

third\_attr = self.combos[2].get()

agg\_method = self.combos[3].get()

agg\_func = None

if agg\_method == "mean":

agg\_func = np.mean

elif agg\_method == "sum":

agg\_func = np.sum

elif agg\_method == "standard deviation":

agg\_func = np.std

self.clear()

self.pivot = self.dataframe.pivot\_table(index=first\_attr, columns=second\_attr, values=third\_attr, fill\_value=0,

aggfunc=agg\_func)

print(self.pivot)

self.report\_type = "TEXT"

self.text\_type = "PIVOT"

self.scrollbar1 = tk.Scrollbar(self.plot\_area\_frame, orient=tk.HORIZONTAL)

self.scrollbar1.pack(side='bottom', fill='x')

self.scrollbar2 = tk.Scrollbar(self.plot\_area\_frame, orient=tk.VERTICAL)

self.scrollbar2.pack(side='right', fill='y')

self.tree = ttk.Treeview(self.plot\_area\_frame, columns=[i for i in self.pivot.keys()], height=50,

xscrollcommand=self.scrollbar1.set, yscrollcommand=self.scrollbar2.set)

self.tree.pack(side='right')

self.to\_file = self.pivot

for i in self.pivot.keys():

self.tree.column(i, anchor=tk.CENTER)

for i in self.pivot.keys():

self.tree.heading(i, text=i)

self.scrollbar1.config(command=self.tree.xview)

self.scrollbar2.config(command=self.tree.yview)

self.tree.config(columns=[i for i in self.pivot.columns])

for i in self.pivot.columns:

self.tree.column(i, anchor=tk.CENTER)

self.tree.heading(i, text=i)

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

[self.tree.insert('', 'end', values=row\_data[1], text=row\_data[0]) for row\_data in

zip(self.pivot.index.tolist(), self.pivot.values.tolist())]

self.additional\_clear.append(self.tree)

self.additional\_clear.append(self.scrollbar1)

self.additional\_clear.append(self.scrollbar2)

def major\_desc\_stats(self, atributes1):

"""Построение таблицы основных описательных статистик"""

for i in self.additional\_clear:

i.destroy()

self.additional\_clear.clear()

atributes = [self.v\_attrs[i] for i in list(atributes1)]

self.index = atributes

self.data = {'Среднее арифметическое': [round(np.mean(self.dataframe[i].values)) for i in atributes],

'Мода': [Counter(np.array(self.dataframe[i].values).flat).most\_common(1)[0][0] for i in atributes],

'Медиана': [np.median(self.dataframe[i].values) for i in atributes],

'Стандартное отклонение': [round(np.std(self.dataframe[i].values)) for i in atributes]}

self.df = pd.DataFrame(self.data, index=self.index)

self.scrollbar1 = tk.Scrollbar(self.plot\_area\_frame, orient=tk.HORIZONTAL)

self.scrollbar1.pack(side='bottom', fill='x')

self.scrollbar2 = tk.Scrollbar(self.plot\_area\_frame, orient=tk.VERTICAL)

self.scrollbar2.pack(side='right', fill='y')

self.tree = ttk.Treeview(self.plot\_area\_frame, columns=[i for i in self.df.columns], height=35,

xscrollcommand=self.scrollbar1.set, yscrollcommand=self.scrollbar2.set)

self.tree.pack(side='right')

for i in self.df.columns:

self.tree.column(i, anchor=tk.CENTER)

for i in self.df.columns:

self.tree.heading(i, text=i)

self.tree.heading("#0", text="Атрибут")

self.scrollbar1.config(command=self.tree.xview)

self.scrollbar2.config(command=self.tree.yview)

self.additional\_clear.append(self.tree)

self.additional\_clear.append(self.scrollbar1)

self.additional\_clear.append(self.scrollbar2)

[self.tree.insert('', 'end', values=row\_data[1], text=row\_data[0]) for row\_data in

zip(self.df.index.tolist(), self.df.values.tolist())]

self.text\_type = "MDS"

self.report\_type = "TEXT"

self.to\_file = self.df

def open\_save\_report(self):

"""Сохранение отчета"""

SaveReport(self.to\_file, self.report\_type, self.text\_type)

class SaveReport(tk.Toplevel):

"""

Окно сохранения отчета

Автор: Балескин

"""

def \_\_init\_\_(self, to\_file, report\_type, text\_type="NONE"):

""""""

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

# self.plottype1 = plottype

# self.figure1 = figure

# self.text\_type1 = text\_type

# self.pivot1 = pivot

self.to\_file = to\_file

self.report\_type = report\_type

self.text\_type = text\_type

self.view = app

self.init\_child()

def init\_child(self):

""""""

self.title('Сохранение отчета')

self.geometry('300x200+400+300')

self.resizable(False, False)

self.label\_name1 = tk.Label(self, text='Введите название файла (без расширения):')

self.label\_name1.pack(side='top', pady=7)

self.entry = ttk.Entry(self, width=30)

self.entry.pack(side='top', pady=7)

self.label\_name1 = tk.Label(self, text='Выберите расширение:')

self.label\_name1.pack(side='top', pady=7)

if self.report\_type == 'GRAPH':

self.combobox = ttk.Combobox(self, values=['PDF', 'PNG', 'JPEG'])

elif self.report\_type == 'TEXT':

self.combobox = ttk.Combobox(self, values=['xlsx'])

self.combobox.pack(side='top', pady=7)

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

self.btn\_cancel.place(x=210, y=160)

self.btn\_ok = ttk.Button(self, text='Сохранить', command=self.save)

self.btn\_ok.place(x=130, y=160)

# self.btn\_ok.bind('<Button-1>', lambda event: self.save)

self.grab\_set()

self.focus\_set()

def save(self, event=None):

"""сохранение"""

self.filename = self.entry.get()

self.extension = self.combobox.get()

if self.report\_type == 'GRAPH':

f = open(

os.path.dirname(os.path.realpath(\_\_file\_\_)) + "\\..\\Graphics\\" + self.filename + '.' + self.extension,

'w')

f.close()

self.to\_file.savefig(fname=os.path.dirname(

os.path.realpath(\_\_file\_\_)) + "\\..\\Graphics\\" + self.filename + '.' + self.extension)

elif self.report\_type == 'TEXT':

sheet\_name = ''

if self.text\_type == 'PIVOT':

sheet\_name = 'pivot\_table'

else:

sheet\_name = 'main\_statistics'

f = open(

os.path.dirname(os.path.realpath(\_\_file\_\_)) + "\\..\\Output\\" + self.filename + '.' + self.extension,

'w')

f.close()

self.to\_file.to\_excel(

os.path.dirname(os.path.realpath(\_\_file\_\_)) + "\\..\\Output\\" + self.filename + '.' + self.extension,

sheet\_name=sheet\_name)

self.destroy()

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

root = tk.Tk()

cp = ConfigurationParser("config.ini")

config = cp.parse()

main\_dbs = config["paths"]

dp = DatabaseParser(list(main\_dbs.items.keys()), main\_dbs.items, hints=config["hints"].items)

dp.parse("DISPLAY", ["-e", None], [None, None], None)

atributes = dp.working\_db.get\_db().columns

app = Main(root, atributes)

app.pack()

root.title("База данных CPU")

root.state('zoomed')

root.geometry("650x345+300+200")

root.mainloop()

**Файл DataBase.py находится в каталоге Library**

"""

БД

Автор: Балескин

"""

import os

from pandas import DataFrame

from pandas import Series

from pandas import read\_csv

class DataBase:

"""

Автор: Балескин

Содержит pandas.DataFrame и замещает его, с целью инкапсуляции лишнего функционала там, где он не нужен

"""

def \_\_init\_\_(self, db=None, name=None):

"""Конструктор от DataFrame и имени БД Автор - Балескин"""

if db is None:

self.db = DataFrame()

else:

self.db = DataFrame(db)

if name is None:

self.name = ''

else:

self.name = name

def get\_db(self):

"""Позволяет получить подлежащий DataFrame Автор - Балескин"""

return self.db

def get\_attr\_names(self):

"""Позволяет получить список атрибутов БД. Автор - Балескин"""

d = [i for i in self.db.keys()]

return d

def append\_object(self, index, column\_data, object\_data):

"""Позволяет добавить объект. Автор - Балескин. Вход - данные об объекте"""

self.db = self.db.append(Series(object\_data, index=column\_data, name=index))

def append\_attribute(self, attr\_name, attr\_values):

"""Позволяет добавить атрибут. Автор - Балескин. Вход - данные об атрибуте"""

self.db[attr\_name] = attr\_values

def append\_attributes(self, attr\_names, db):

"""Позволяет добавить множество атрибутов. Автор - Балескин. Вход - данные об атрибутах"""

for i in attr\_names:

self.append\_attribute(i, db.get\_attributes([i]).get\_db())

def delete\_objects(self, object\_keys):

"""Позволяет удалить объекты. Автор - Балескин. Вход - ключи объектов"""

self.db = self.db.drop(object\_keys, axis=0)

def delete\_attribute(self, attr\_name):

"""Позволяет удалить атрибут. Автор - Балескин. Вход - имя атрибута"""

if attr\_name is not None:

self.db = self.db.drop([attr\_name], axis=1)

def get\_value(self, key, attr\_name):

"""Позволяет получить значение ячейки. Автор Балескин. Вход - ключ объекта и имя атрибута"""

return self.db.at[key, attr\_name]

def change\_value(self, key, attr\_name, new\_value):

"""Позволяет изменить значение ячейки. Автор Балескин. Вход - ключ объекта и имя атрибута, новое значение

возвращает старое значение"""

temp = self.get\_value(key, attr\_name)

self.db[attr\_name].loc[key] = new\_value

return temp

def rename(self, mapper, axis):

"""Позволяет переименовать объект или атрибут Автор - Балескин"""

self.db = self.db.rename(mapper, axis=axis)

def get\_objects(self, keys):

"""Позволяет получить множество объектов по их ключам Автор - Балескин"""

k = self.db.loc[keys, :]

return DataBase(k)

def get\_objects\_exclusive(self, keys):

"""Позволяет получить все объекты кроме данных. Задаются объекты в виде ключей Автор - Балескин"""

k = self.db.drop(keys, axis=0)

return DataBase(k)

def get\_attributes(self, attr\_names):

"""Позволяет получить множество атрибутов по их ключам Автор - Балескин"""

k = self.db[attr\_names]

return DataBase(k)

def get\_attributes\_exclusive(self, attr\_names):

"""Позволяет получить все атрибуты кроме данных. Задаются атрибуты по их названиям Автор - Балескин"""

k = self.db.drop(attr\_names, axis=1)

return DataBase(k)

def get\_part(self, keys, attr\_names):

"""Позволяет получить подмножество БД по указанным ключам объектов и именам атрибутов Автор - Балескин"""

if attr\_names is None:

return self.get\_objects(keys)

if keys is None:

return self.get\_attributes(attr\_names)

k = self.db[attr\_names].loc[keys, :]

return DataBase(k)

def get\_part\_exclusive(self, keys, attr\_names):

"""Позволяет получить подмножество БД по ключам объектов и именам атрибутов, исключая указанные Автор - Балескин"""

k = self.db.drop(attr\_names, axis=1)

a = k.drop(keys, axis=0)

return DataBase(a)

def get\_name(self):

"""Позволяет получить имя БД Автор - Балескин"""

return self.name

def join(self, other, on, how):

"""Позволяет слить две БД Автор - Балескин"""

db = DataFrame(self.db)

if self.db.empty:

return DataBase(other.get\_db(), name=self.name)

return DataBase(db.join(other.get\_db(), on, how, lsuffix="\_caller", rsuffix="\_callee"))

def empty(self):

"""Позволяет проверить, не пуста ли БД Автор - Балескин"""

return self.db.empty

def store(self, filename):

"""Позволяет сохранить БД в файл Автор - Балескин"""

path = os.path.dirname(os.path.realpath(\_\_file\_\_)) + '\\..\\Data\\'

self.db.to\_csv(path + filename + '.csv', index=True)

def read(self, filename):

"""Позвоялет считать БД из файла"""

from pathlib import Path

path = os.path.dirname(os.path.realpath(\_\_file\_\_)) + '\\..\\Data\\'

path += filename + '.csv'

file = Path(path)

if not file.is\_file():

raise RuntimeError("Not a file")

self.db = read\_csv(path, index\_col=0)

# print(self.db)

**Файл ConfigurationParser.py находится в каталоге Library**

"""Автор: Балескин"""

import os

class Block:

"""

Хранит один блок файла ini

Автор: Балескин

"""

def \_\_init\_\_(self, block\_name):

""""""

self.name = block\_name

self.items = dict()

def add\_item(self, item\_name, item\_value):

self.items[item\_name] = item\_value

class ConfigurationParser:

"""Автор: Балескин"""

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

""""""

self.path = path

self.blocks = dict()

def parse(self):

""""""

# print(os.path.dirname(os.path.realpath(\_\_file\_\_)))

with open(os.path.dirname(os.path.realpath(\_\_file\_\_)) + '\\..\\Scripts\\' + self.path, 'r') as f:

block\_name = ""

strs = f.readlines()

for line in strs:

if line[0] == '[':

block\_name = line[1:-2]

self.blocks[block\_name] = Block(block\_name)

elif line != "":

name, val = line.split('=')

val = val[:-1] if val[-1] == '\n' else val

self.blocks[block\_name].add\_item(name, val)

return self.blocks

**Файл DatabaseParser.py находится в каталоге Library/database\_interaction**

"""

Содержит обработчик запросов к базе данных.

Задача: искоренить влияние Pandas (по возможности) из потока данных. Поток данных не должен от него зависеть.

Вполне возможно (и так действительно может получиться!), обработчики и GUI будут частично зависеть от Pandas,

(используем Pandas как обертку для MatPlotLib) но потоку не должно быть никакого дела до передаваемых по нему объектов.

Author: Vitaly(Admin)

Автор: Балескин

"""

from Work.Library.DataBase import DataBase

from Work.Library.DataFlow.DBInterface import DBInterface

class DatabaseParser(DBInterface):

"""

Обработчик БД, множества БД, сессионной БД и запросов к оным

Автор: Балескин

"""

def \_\_init\_\_(self, names, paths, hints=None):

"""

Конструктор. Добывает БД из файлов csv. Вход - имена справочников, словарь имя справочника:путь к нему, подсказки для слияния

и помещает их в словарь отношение:БД Автор: Балескин

"""

assert len(names) == len(paths)

self.its\_dbs = {i: DataBase() for i in names}

if hints is not None:

self.hints = {i: hints[i] for i in names}

for i in names:

self.its\_dbs[i].read(paths[i])

self.attr\_names = {names[i]: self.its\_dbs[names[i]].get\_attr\_names() for i in range(len(names))}

self.working\_db = DataBase()

def inclusive\_attr\_display(self, attr\_names):

"""

отобразить все атрибуты из списка в сессионной бд Автор - Балескин

вход - имена атрибутов

"""

excl\_attr\_names = []

for attrs in self.attr\_names.values():

for attr in attrs:

if attr not in attr\_names:

excl\_attr\_names.append(attr)

self.exclusive\_attr\_display(excl\_attr\_names)

def inclusive\_obj\_display(self, obj\_names):

"""Отобразить все объекты из списка на сессионную БД Автор - Балескин

вхд - имена объектов"""

self.working\_db = self.working\_db.get\_objects(obj\_names)

def exclusive\_attr\_display(self, attr\_names):

"""Отобразить все атрибуты кроме указанных в списке Автор - Балескин

Вход - имена атрибутов"""

self.working\_db = DataBase()

for (name, db) in self.its\_dbs.items():

a = self.working\_db.join(db, on=self.hints[name] if self.hints[name] != '-' else None, how='left')

self.working\_db = a

if attr\_names is not None:

for attr in attr\_names:

self.working\_db.delete\_attribute(attr)

self.working\_db = DataBase(self.working\_db.get\_db().drop\_duplicates())

self.working\_db = DataBase(self.working\_db.get\_db().groupby(self.working\_db.get\_db().index).first())

def exclusive\_obj\_display(self, obj\_names):

"""Отобразить все объекты, кроме указанных Автр - Балескин

Вход - список объектов"""

self.working\_db.delete\_objects(obj\_names)

def parse(self, q\_type, \*pargs):

"""

Автор - Балескин

Метод, отвечающий за обработку.

Выполняет выбор типа операции в зависимости от аргументов (в частности, первого) и делегирует частным случаям.

Первый аргумент - тип запроса

Второй - условия

Итого:

query -> query\_type args

query\_type -> DISPLAY | ADD | DROP | APPEND | DELETE | STORE | CHANGE | RENAME | LOAD

args -> attr\_list obj\_list conditions |

names path\_names hints|

names |

\_names \_names values name|

\_names \_names values|

pathname|

obj\_name attr\_name value|

dictionary, axis|

name

names, path\_names, \_names, name - строки (или списки строк!) (возможны пустые строки)

attr\_list -> key \_names | NONE

obj\_list -> key \_names | NONE

key -> NONE | -e | -i (Отсутствие ключа = Отсутствие аргумента -> тогда под условие попадает все!) i - включая; e - исключая

conditions - процессоры boolean(Database, obj\_name, attr\_name)

"""

if q\_type == 'DISPLAY':

attr\_key = pargs[0][0]

attr\_names = pargs[0][1:]

obj\_key = pargs[1][0]

obj\_names = pargs[1][1:]

conditions = pargs[2]

if self.working\_db.empty():

self.working\_db = DataBase(name='DATA')

if attr\_key is not None:

if attr\_key == '-i':

self.inclusive\_attr\_display(attr\_names)

elif attr\_key == '-e':

self.exclusive\_attr\_display(attr\_names)

if obj\_key is not None:

if obj\_key == '-i':

self.inclusive\_obj\_display(obj\_names)

elif obj\_key == '-e':

self.exclusive\_obj\_display(obj\_names)

if conditions is not None:

for cond in conditions:

cond.\_process(self.working\_db)

elif q\_type == 'ADD':

names = pargs[0]

path\_names = pargs[1]

hints = pargs[2]

for (name, hint) in hints.items():

self.hints[name] = hint

for i in range(len(names)):

db = DataBase()

db.read(path\_names[i])

self.its\_dbs[names[i]] = db

self.attr\_names = {i: self.its\_dbs[i].get\_attr\_names() for i in self.its\_dbs.keys()}

elif q\_type == 'DROP':

names = pargs[0]

for name in names:

del self.its\_dbs[name]

self.attr\_names = {i: self.its\_dbs[i].get\_attr\_names() for i in self.its\_dbs.keys()}

elif q\_type == 'APPEND':

attr\_names = pargs[0]

obj\_names = pargs[1]

obj\_values = pargs[2]

name = pargs[3]

if name is not None:

self.working\_db.join(self.its\_dbs[name].get\_attributes(attr\_names), on=None, how='inner')

else:

global\_failure = False

for attr in attr\_names:

failure = True

for n, attrs in self.attr\_names.items():

if attr in attrs:

self.working\_db.join(self.its\_dbs[n].get\_attributes([attr]), on=None, how='inner')

failure = False

break

if failure:

self.working\_db.append\_attribute(attr, [None for i in self.working\_db.get\_db().values])

global\_failure = True

objects = dict(zip(obj\_names, obj\_values))

for o\_name, values in objects.items():

self.working\_db.append\_object(o\_name, list(values.keys()), list(values.values()))

elif q\_type == 'DELETE':

attr\_names = pargs[0]

obj\_names = pargs[1]

self.working\_db.delete\_objects(obj\_names)

for attr in attr\_names:

self.working\_db.delete\_attribute(attr);

elif q\_type == 'STORE':

path\_name = pargs[0]

self.working\_db.store(path\_name)

elif q\_type == 'CHANGE':

attr\_name = pargs[0]

obj\_name = pargs[1]

value = pargs[2]

self.working\_db.change\_value(obj\_name, attr\_name, value)

elif q\_type == 'RENAME':

mapper = pargs[0]

axis = pargs[1]

self.working\_db.rename(mapper, axis)

elif q\_type == 'LOAD':

self.working\_db.read(pargs[0])

pass

**Файл ConfigurationParser.py находится в каталоге Library**

"""Автор: Балескин"""

import os

class Block:

"""

Хранит один блок файла ini

Автор: Балескин

"""

def \_\_init\_\_(self, block\_name):

"""Конструктор от имени блока Автор-Балескин"""

self.name = block\_name

self.items = dict()

def add\_item(self, item\_name, item\_value):

"""Добавление элемента Автор - Балескин"""

self.items[item\_name] = item\_value

class ConfigurationParser:

"""Обработчик файла конфигурации Автор: Балескин"""

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

"""Конструктор от пути к файлу Автор-Балескин"""

self.path = path

self.blocks = dict()

def parse(self):

"""Обработка. Возвращает элементы файла Автор Балескин"""

# print(os.path.dirname(os.path.realpath(\_\_file\_\_)))

with open(os.path.dirname(os.path.realpath(\_\_file\_\_)) + '\\..\\Scripts\\' + self.path, 'r') as f:

block\_name = ""

strs = f.readlines()

for line in strs:

if line[0] == '[':

block\_name = line[1:-2]

self.blocks[block\_name] = Block(block\_name)

elif line != "":

name, val = line.split('=')

val = val[:-1] if val[-1] == '\n' else val

self.blocks[block\_name].add\_item(name, val)

return self.blocks

# **Технические требования**

32- или 64-битная операционная система Windows, Linux, MacOS, на которую возможна установка интерпретатора python 3.7(<https://www.python.org/>). Также требуется минимум 1,5 ГГц тактовой частоты процессора и разрешение экрана 1280x720 и выше.