ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ЮГОРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ДОКУМЕНТАЦИЯ

к программному продукту «Цифровой двойник нефтегазоконденсатного месторождения»

на конкурс

Хакатон «Цифровые двойники в энергетике»

Проект выполнен командой ЮГУ:

Руководитель: Ткаченко Всеволод Андреевич

Члены команды:

Солодянкин Матвей Сергеевич магистрант Электроэнергетики и электротехники ЭЭ41м

Дудоров Егор Сергеевич бакалавр Программной Инженерии ПИ41Б

Полуянов Андрей Сергеевич бакалавр Программной Инженерии ПИ41Б)

Гарифзянов Эмиль Эдуардович бакалавр Программной Инженерии ПИ41Б

Кочетков Михаил Антонович бакалавр Электроэнергетики и Электротехники 2821б

Оглавление

[1 Общие сведения о программе 3](#_Toc195173389)

[2 Тепловая часть 4](#_Toc195173390)

[3 Электрическая часть 5](#_Toc195173391)

[4 Руководство пользователя 6](#_Toc195173392)

[5 Листинг программы 10](#_Toc195173393)

# Общие сведения о программе

Программа создана для визуализации работы энергетической системы промышленного объекта.

Программный продукт реализует:

* Отображение состояния ГТУ и котлов-утилизаторов,
* Загрузку данных из Excel файлов,
* Подсчёт потребности тепла в зависимости от температуры наружного воздуха,
* Определение стоимости потребления газа,
* Моделирование работы ГТУ и котлов с учётом заданных условий.

Программа написана на языке программирования общего назначения Python 3 c применением библиотек:

* Tkinter – графический интерфейс,
* Pandas – работа с Exel,
* PIL – работа с изображениями,
* NumPy – математические расчеты,
* Dateutil – работа с датами.

# Тепловая часть

Математическая модель тепловой части энергосистемы основана на представлении каждого котла-утилизатора как объекта с набором параметров. Количество котлов определяется по максимальной требуемой тепловой мощности.

В качестве потребляемой тепловой мощности принимается величина равная сумме тепловой мощности всех потребителей сети, которая аппроксимируется по температуре воздуха. Расчет выработки тепла начинается при понижении температуры воздуха ниже +8 градусов Цельсия.

После этого определяется количество котлов, которое необходимо для покрытия тепловой потребности, вместе с этим находится значение загрузки котлов, которое определяется пропорционально их количеству.

Производится запуск расчетного количества котлов. Если на данном этапе расчета какое-то количество котлов уже было запущено и это число больше необходимого, то лишнее количество отключается.

Производится загрузка котлов по раннее определенному значению.

После загрузки котлов рассчитывается стоимость потребляемого газа всех котлов за час.

# Электрическая часть

Математическая модель электрической части энергосистемы основана на представлении каждой ГТУ как объекта с набором параметров.

Количество ГТУ определяется по максимальной потребляемой электрической мощности.

В качестве максимального значения мощности принимается величина равная сумме мощностей потребителей и потерь мощности в элементах сети (линия электропередачи). Для определения потерь использован матричный метод расчета установившегося режима электрической сети, определены токи в линия, через сопротивление и ток в линиях определены потери мощности.

После определения необходимого количества ГТУ определяется их загрузка, а по величине загрузки с использованием аппроксимации определяется отношение моточасов к часам реального времени (МЧкРЧ).

Далее на сутки по отношению МЧкРЧ и затратам моточасов на пуск и останов ГТУ определяется возможная наработка. Если с учетом суточной наработки ГТУ выходит за предел моточасов до ТО или КР, то ГТУ отключается и переводится в соответствующее состояние со сбросом времени наработки до следующего ТО или КР, активируется отсчет в сутках до выхода ГТУ из этого состояния.

При недостатке ГТУ до необходимого количества выбираются ГТУ, которые имеют меньшее время до КР. При большом количестве ГТУ отключаются те, которые имеют большее время до КР.

За перевод ГТУ в состояние ТО или КР к общим затратам добавляется соответствующая стоимость.

# Руководство пользователя

Для начала работы необходимо подгрузить файлы с исходными данными, нажав на кнопку «Загрузить данные» в правом нижнем углу экрана.

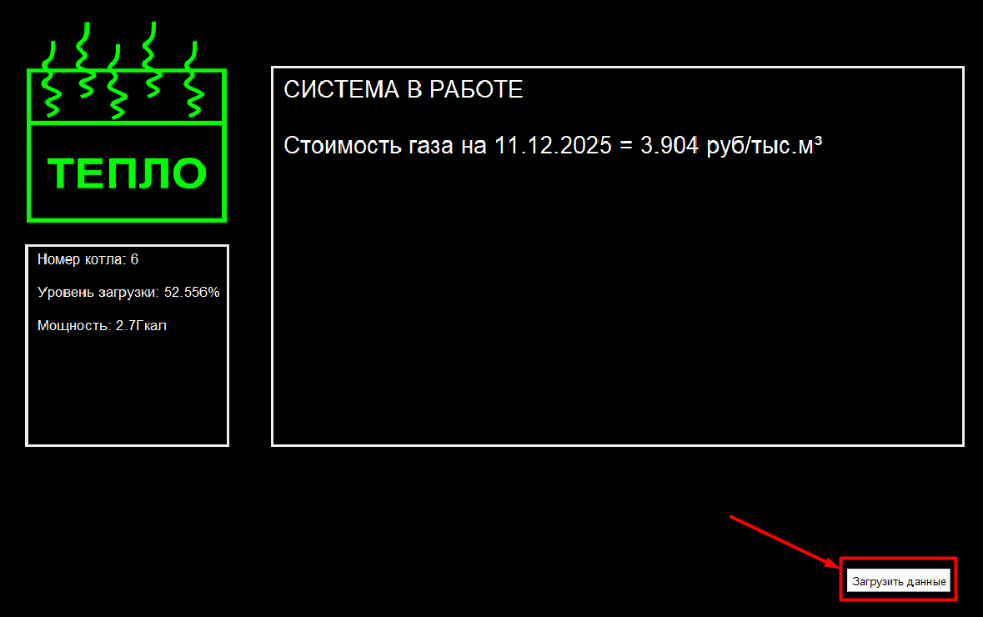


Рисунок 1

В левом нижнем углу основного располагаются кнопки управления датой. Кнопками: «Предыдущий месяц», «Предыдущий день», «Следующий день», «Следующий месяц» можно выбирать необходимую дату, которая отображается с права от кнопок.



Рисунок 2 – Кнопки управления

Каждая иконка (рис. 3, 4) на экране символизирует отдельные установки со своими независимыми друг от друга параметрами.

На рисунке 3 изображена иконка, которая обозначает одну из девяти ГТУ.

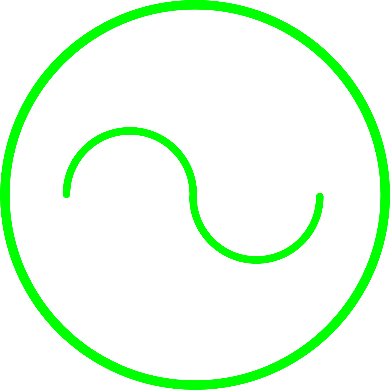


Рисунок 3 – ГТУ (в работе)

На рисунке 4 изображена иконка, которая обозначает один из шести котлов-утилизаторов.

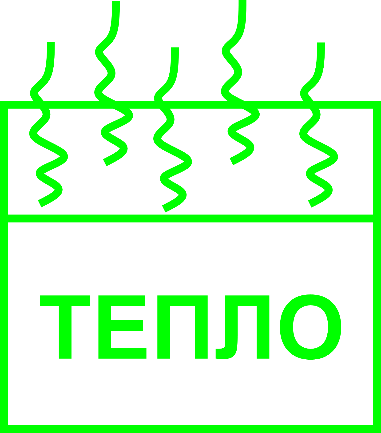


Рисунок 4 – Котел-утилизатор (в работе)

Иконки (рис. 3, 4) являются интерактивными объектами и изменяют свой цвет в зависимости от состояния установки в текущий момент времени.

Так, зеленый цвет установки (рис 3, 4) указывает на исправно работающее состояние установки. В этом состоянии установка исполняет полезную работу (вырабатывает электроэнергию, тепло).

Красный цвет (рис. 5, 6) указывает на то, что установка в данный момент времени отключена. Отключение любой установки происходит из-за того, что в данный момент времени держать ее во включенном состоянии нецелесообразно.

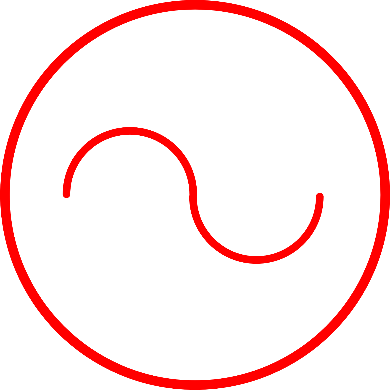


Рисунок 5 – ГТУ (отключена)

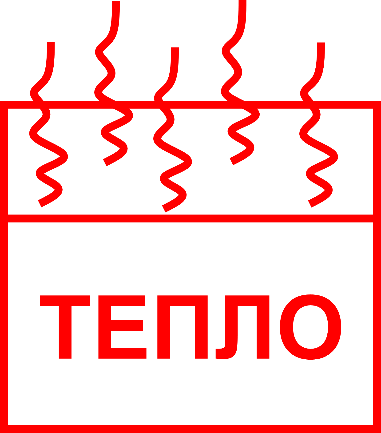


Рисунок 6 – Котел (отключен)

Синий цвет (рис. 7) уникален для ГТУ и указывает на то, что в данный момент времени установка находится в состоянии ремонта (ТО или КР).

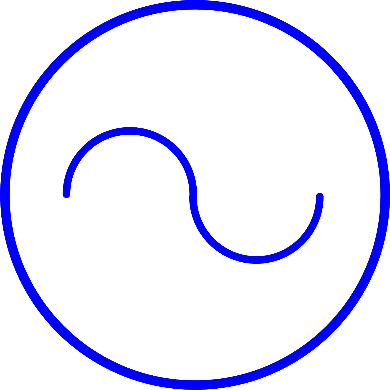


Рисунок 7 – ГТУ (в ремонте)

Под каждой иконкой в прямоугольном окне (рис. 8) выводятся основные параметры установок.

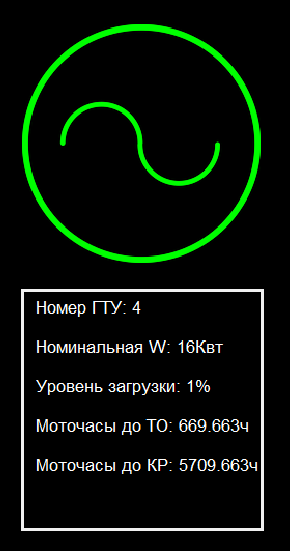


Рисунок 8 – Окно с параметров установки

Параметры для ГТУ:

* Номер ГТУ
* Номинальная мощность, МВт
* Уровень загрузки, %
* Моточасы до ТО, мч
* Моточасы до КР, мч

Параметры для котлов-утилизаторов:

* Номер котла
* Номинальная мощность, Гкал
* Уровень загрузки, %

В правой части основного окна (рис. 9) располагается окно вывода системных параметров модели.

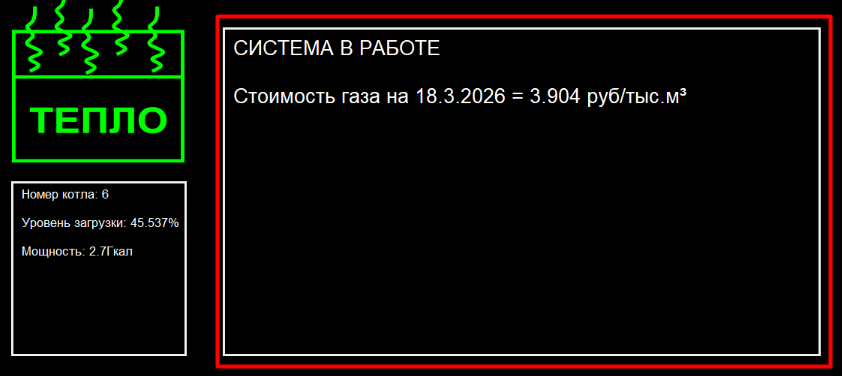


Рисунок 9 – Окно вывода системных данных

# Листинг программы

|  |
| --- |
| 1. *from* PIL *import* Image 2. *import* ctypes *#Подключаем типы из С/С++* 3. *from* datetime *import* datetime, timedelta 4. *from* dateutil.relativedelta *import* relativedelta *#изменение месяца pip install python-dateutil* 5. *import* math 6. *import* numpy 7. *import* pandas *as* pd 8. *import* random 9. *from* tkinter *import* \* 10. *from* tkinter.filedialog *import* askopenfilenames 11. *from* tkinter.messagebox *import* showinfo, showerror 12. *from* PIL *import* Image, ImageTk 13. *# Глобальные переменные для моточасов из Excel* 14. initial\_to\_hours = [] 15. initial\_kr\_hours = [] 16. is\_fullscreen = False 17. root = Tk() 18. root.title('Визуализация системы') 19. root['bg'] = 'black' 20. *# Получаем ширину и высоту экрана* 21. *# Полный размер экрана (вместе с панелью задач)* 22. user32 = ctypes.windll.user32 23. user32.SetProcessDPIAware()  *# Учитываем масштабирование Windows* 24. screen\_width = user32.GetSystemMetrics(0)  *# Ширина экрана* 25. screen\_height = user32.GetSystemMetrics(1)  *# Высота экрана* 26. *# Получаем размеры рабочей области экрана (без панели задач)* 27. *# Используем системную функцию Windows SystemParametersInfoW* 28. spi\_getworkarea = 48  *# Константа для получения рабочей области* 29. rect = ctypes.wintypes.RECT()  *# Структура для хранения координат рабочей области* 30. *# Заполняем структуру rect размерами рабочей области* 31. ctypes.windll.user32.SystemParametersInfoW(spi\_getworkarea, 0, ctypes.byref(rect), 0) 32. *# Рассчитываем ширину и высоту рабочей области* 33. work\_width = rect.right - rect.left 34. work\_height = rect.bottom - rect.top 35. *# Устанавливаем размеры и положение окна в пределах рабочей области* 36. *# {ширина}x{высота}+{смещение по X}+{смещение по Y}* 37. root.geometry(f'{work\_width}x{work\_height}+{rect.left}+{rect.top}') 38. w = work\_width 39. h = work\_height *#Переопределение рабочей области* 40. def *open\_excel\_files*(): 41. filepaths = askopenfilenames(filetypes=[("Excel files", "\*.xlsx")]) 42. *if* not filepaths: 43. *return* 44. *try*: 45. *global* energy\_usage\_data, gtu\_data, gas\_data, months, gas\_prices 46. *global* m\_hours, m\_to\_hours, m\_kr\_hours  *# Объявляем глобальными* 47. gtu\_path = filepaths[0] 48. gas\_path = filepaths[1] 49. gtu\_data = pd.read\_excel(gtu\_path) 50. gas\_data = pd.read\_excel(gas\_path, sheet\_name='Лист1', header=None) 51. *# Обработка газовых цен* 52. last\_col = gas\_data.loc[2, 1:].last\_valid\_index() 53. months = gas\_data.loc[0, 1:last\_col].to\_numpy() 54. gas\_prices = gas\_data.loc[2, 1:last\_col].to\_numpy() 55. *# Расчет моточасов только при загрузке файлов* 56. last\_col\_index = gtu\_data.columns[-1] 57. m\_hours = gtu\_data.loc[3:, last\_col\_index].dropna().to\_numpy(dtype=float) 58. m\_to\_hours = numpy.where(m\_hours > 1500, m\_hours % 1500, m\_hours) 59. m\_kr\_hours = numpy.where(m\_hours > 10000, m\_hours % 10000, m\_hours) 60. m\_hours = numpy.round(m\_hours, 3) 61. *global* initial\_to\_hours, initial\_kr\_hours, gtes 62. initial\_to\_hours = m\_to\_hours.copy() 63. initial\_kr\_hours = m\_kr\_hours.copy() 64. *# Создаём список объектов GTU по числу данных* 65. gtes = [GTU(i) *for* i *in* range(len(m\_hours))] 66. *# Присваиваем каждому GTU его начальные данные* 67. *for* idx, gtu *in* enumerate(gtes): 68. gtu.to = initial\_to\_hours[idx] 69. gtu.kr = initial\_kr\_hours[idx] 70. print("Файлы успешно загружены!") 71. showinfo("Успех!", "Файлы успешно загружены!") 72. set\_status\_message(f"СИСТЕМА В РАБОТЕ\n\n{price\_calc()}") 73. boilers\_initialization() 74. gtu\_initialization() 75. *except* Exception *as* e: 76. showerror("Ошибка!", f"Ошибка при загрузке файлов:\n{e}") 77. *# Читаем данные* 78. current\_date = datetime.now().date() 79. def *price\_calc*(): 80. *global* price 81. price = 0  *# Сначала обнуляем цену газа* 83. *# Текущая дата* 84. current\_day = current\_date.day 85. current\_month = current\_date.month 86. current\_year = current\_date.year 87. *if* 'gas\_prices' not in globals(): 88. print("Цены на газ ещё не загружены.") 89. *return* 90. *# Проверяем год и рассчитываем цену только если есть данные* 91. *if* current\_year == 2024: 92. *if* current\_month <= len(gas\_prices): 93. price = gas\_prices[current\_month - 1]/1000 94. *elif* current\_year == 2025: 95. price = gas\_prices[-1]/1000 96. *elif* current\_year == 2026: 97. price = gas\_prices[-1]/1000 98. *else*: 99. price = 0  *# Нет данных на этот год* 100. *# Вывод в терминал* 101. *if* price != 0: 102. *return*(f'Стоимость газа на {current\_day}.{current\_month}.{current\_year} = {price:.3f} руб/тыс.м³') 103. *else*: 104. *return*(f'Нет данных для расчета стоимости газа за {current\_month}.{current\_year}') 106. def *fullscreen*(event): 107. *global* is\_fullscreen 108. is\_fullscreen = not is\_fullscreen 109. root.attributes('-fullscreen', is\_fullscreen) 110. *# Загрузка и масштабирование изображения* 111. def *load\_scaled\_image*(path, size): 112. img = Image.open(path) 113. img = img.resize((int(size), int(size)), Image.LANCZOS) 114. *return* ImageTk.PhotoImage(img) 115. def *update\_label*(): 116. date\_label.config(text=current\_date.strftime("%d.%m.%Y"), fg='white', bg='black', font=('Arial', 14, 'bold')) 117. def *next\_date*(): 118. *global* current\_date 119. current\_date += timedelta(days=1) 120. update\_label() 121. set\_status\_message(f"СИСТЕМА В РАБОТЕ\n\n{price\_calc()}") 122. boilers\_initialization() 123. gtu\_initialization() 124. def *previous\_date*(): 125. *global* current\_date 126. current\_date -= timedelta(days=1) 127. update\_label() 128. set\_status\_message(f"СИСТЕМА В РАБОТЕ\n\n{price\_calc()}") 129. boilers\_initialization() 130. gtu\_initialization() 131. def *next\_month*(): 132. *global* current\_date 133. current\_date += relativedelta(months=1) 134. update\_label() 135. set\_status\_message(f"СИСТЕМА В РАБОТЕ\n\n{price\_calc()}") 136. boilers\_initialization() 137. gtu\_initialization() 138. def *previous\_month*(): 139. *global* current\_date 140. current\_date -= relativedelta(months=1) 141. update\_label() 142. set\_status\_message(f"СИСТЕМА В РАБОТЕ\n\n{price\_calc()}") 143. boilers\_initialization() 144. gtu\_initialization() 145. class UtilizationBoiler: 146. """Котел утилизатор КВ-ГМ-3,15-95. 148. Внутренние атрибуты: 149. num\_boiler - номер утилизатора 150. pwr - номинальная мощность, Гкал 151. kpd - КПД 152. load - уровень загрузки, % 153. status - True/False (вкл/выкл) 154. """ 156. def \_\_init\_\_(self, num\_boiler): 157. self.num = num\_boiler 158. self.pwr = 2.7 159. self.kpd = 0.935 160. self.load = 0 161. self.status = False 162. def *start*(self): 163. """Запуск котла""" 164. *if* not self.status: 165. self.status = True 166. *return* f'Котел-утилизатор {self.num} запущен.' 167. *return* f'Котел-утилизатор {self.num} уже запущен.' 168. def *stop*(self): 169. """Останов котла""" 170. *if* self.status: 171. self.status = False 172. *return* f'Котел-утилизатор {self.num} остановлен.' 174. *return* f'Котел-утилизатор {self.num} уже остановлен.' 175. def *load\_b*(self, load\_boil): 176. """Изменение загрузки котла""" 177. self.load = load\_boil 179. *return* f'Уровень загрузки котла {self.num}: {self.load} %' 181. def *heat\_otpt*(self): 182. """генерируемая тепловая мощность от процента загрузки котла, Гкал 183. load - загрузка котла, % """ 184. *if* self.load > 100: 185. self.load = 100 186. *elif* self.load < 0: 187. self.load = 0 189. heat = 0 190. *if* self.status: 191. heat = float(format( 192. (self.pwr \* self.load/100), '.3f' 193. ) 194. ) 195. *return* heat 197. def *gas\_cons*(self): 198. """Потребление газа котлом, м3/ч""" 199. cons = 0 200. gas\_cal\_val = 0.01075 *# теплотворная способность газа, Гкал/м3* 202. *if* self.status: 203. cons = float(format( 204. ((self.heat\_otpt()/gas\_cal\_val)/self.kpd), '.3f' 205. ) 206. ) 207. *return* cons 208. def \_\_str\_\_(self): 209. *return* f'''Номер котла: {self.num} 210. Номинальная мощность: {self.pwr} 211. Уровень загрузки: {self.load} % / {self.heat\_otpt()} Гкал 212. Состояние вкл/выкл: {self.status} 213. ''' 214. last\_5\_temps = [None, None, None, None, None] 215. def *heat\_from\_temp*(temp): 216. """ 217. Функция потребности тепла в зависимости от отрицательной температуры воздуха 218. temp - температура воздуха 219. Функция должна срабатывать, когда среднесуточная температура 220. на улице держится ниже +8 °C в течение 5 дней подряд 221. """ 222. heat\_need = 0 223. *global* last\_5\_temps 224. last\_5\_temps.append(temp) 225. *if* len(last\_5\_temps) > 5: 226. last\_5\_temps.pop(0) 227. *if* None not in last\_5\_temps and all(t < 8 *for* t *in* last\_5\_temps): 228. *# кубическая регрессия от x: 0 -10 -24 -38 -48; y: 1.126 2.345 5.63 8.914 11.26;* 229. heat\_need = float(format( 230. (0.00006180 \* temp \*\* 3 + 0.00559107 \* temp \*\* 2 - 0.08512969 \* temp + 1.09309420), '.3f' 231. ) 232. ) 234. *return* heat\_need 235. def *heat\_load\_distribution*(ht\_fr\_dist): 236. """Функция распределяет выработку тепла между работающими котлами. 238. quant\_boiler - количество работающих котлов 239. ht\_fr\_dist - тепло, подлежащее распределению (из выхода ф-ии heat\_from\_temp)""" 240. quant\_boiler = 6 241. load = float(format( 242. (((ht\_fr\_dist / quant\_boiler) / 2.7) \* 100), '.3f' 243. ) 244. ) 245. *if* load < 45.0: 246. *while* load < 45.0: 247. quant\_boiler = quant\_boiler - 1 249. *if* quant\_boiler != 1: 250. load = float(format( 251. (((ht\_fr\_dist / quant\_boiler) / 2.7) \* 100), '.3f' 252. ) 253. ) 254. *else*: 255. *break* 256. *if* ht\_fr\_dist == 0: 257. load = 0 258. quant\_boiler = 0 260. *return* load, quant\_boiler 261. def *heat\_cost*(boilers): 262. """Функция расчета стоимости газа, руб/ч""" 263. *global* price 264. *try*: 265. *# Проверяем, рассчитана ли цена* 266. *if* price is None: 267. *raise* ValueError("Ошибка: цена на газ не определена!") 268. gas\_cons = [boiler.gas\_cons() *for* boiler *in* boilers] 269. cost = sum(gas\_cons) \* price 270. *return* cost 271. *except* ValueError *as* e: 272. print(e) 273. *return* 0  *# Возвращаем 0 если цена не определена* 274. *except* Exception *as* e: 275. print(f"Неизвестная ошибка при расчете стоимости газа: {e}") 276. *return* 0 277. def *temp\_of\_month*(): 278. temp = [-28.1, -27.3, -21.6, -14.9, -5.4, 6.1, 13.7, 10.8, 3.9, -8.3, -20.5, -24.7] 279. *return* temp[current\_date.month-1] 280. def *boilers\_initialization*(): 281. *# Инициализация котлов* 282. boilers = [UtilizationBoiler(i) *for* i *in* range(6)] 284. temp = temp\_of\_month() 285. heat\_need = heat\_from\_temp(temp) *# Потребность в тепле* 286. print(f'Потребность в тепле: {heat\_need} Гкал/ч') 287. load\_val = heat\_load\_distribution( heat\_need )[0] 288. print(f'Загрузка одного котла: {load\_val} %') 289. num\_boilers\_must\_on = heat\_load\_distribution( heat\_need )[1] *# Число котлов, которые должны быть запущены* 290. print(f'Должно быть запущено котлов: {num\_boilers\_must\_on}') 291. *# Вкл нужного количества котлов* 292. n\_boil\_on = 6 - num\_boilers\_must\_on 293. *for* boiler *in* boilers: 294. *if* boiler.num > n\_boil\_on - 1: 295. print(boiler.start()) 296. *if* boiler.num < n\_boil\_on - 1: 297. *if* not boiler.status: 298. print(boiler.stop()) 299. *# Передача нужного значения загрузки котлу* 300. *for* boiler *in* boilers: 301. *if* boiler.status: 302. load = boiler.load\_b(load\_val) 303. print(load) 304. print(f'Стоимость газа: {heat\_cost(boilers)} руб/ч') 305. *# Вывод информации о каждом котле после расчета* 306. *for* boiler *in* boilers: 307. print(boiler) 308. BLR\_info(boiler.num + 1, boiler.load, boiler.pwr, boiler.status) 309. class GTU: 310. def \_\_init\_\_(self, n): 311. self.n = n *# номер ГТУ* 312. self.power = 16 *# номинальная мощность* 313. self.load = 1 *# уровень загрузки (о.е.)* 314. self.to = 1500 *# моточасы до ТО* 315. self.kr = 10000 *# моточасы до КР* 316. self.state = 0 *# состояние 0-выкл/1-вкл/2-ТО или КР* 317. self.service\_time = 0 *# время сервисного обслуживания: 14 дней для ТО и 30 дней для КР* 319. def *stop\_to*(self): 320. self.state = 2 321. self.service\_time = 14 322. self.to = 1500 324. def *stop\_kr*(self): 325. self.state = 2 326. self.service\_time = 30 327. self.to = 1500 328. self.kr = 10000 330. def *stop\_n*(self): 331. self.state = 0 332. self.to = self.to - 2.5 333. self.kr = self.kr - 2.5 335. def *start\_n*(self): 336. self.state = 1 337. self.to = self.to - 2.5 338. self.kr = self.kr - 2.5 340. def \_\_str\_\_(self): 341. *return* f''' 342. номер ГТУ {self.n} 343. номинальная мощность {self.power} 344. уровень загрузки (о.е.) {self.load} 345. моточасы до ТО {self.to} 346. моточасы до КР {self.kr} 347. состояние вкл/выкл {self.state} 348. ''' 350. def *hourly\_production*(load): 351. x = load 352. y = -1.6667\*x\*\*3+3.3333\*x\*x-1.1833\*x+0.5167 *# кубическая регрессия для нахождения отношения мч/ч* 354. *return* y 355. def *get\_power\_loss*(temp, season): 356. base\_node\_voltage = 11 *# напряжение базисного узла, кВ* 357. rated\_voltage = 10 *# номинальное напряжение сети, кВ* 359. node\_type = ['base'] *# тип узла* 360. number\_of\_circuits = [] *# количество цепей, шт.* 362. *for* i *in* range(32): 363. node\_type.append('load') 364. number\_of\_circuits.append(2) 366. r0 = 0.245/(1+0.00403\*(20-temp)) *# скорректированное на текущую температуру погонное сопротивление АС-120/19* 367. x0 = 0.38 369. line\_lengths = [0.5, 3, 3.5, 4, 100, 1, 2, 3, 3.3, 4, 4.5, 5, 5.2, 5.7, 6, 6.5, 7, 7.5, 7.8, 8, 8.5, 9, 9.3, 370. 9.7, 17, 18, 20, 22, 23, 24, 25, 0.3] *# длины линий* 372. linear\_active\_resistance = [ll\*r0 *for* ll *in* line\_lengths] *# активное сопротивление ветвей, Ом* 373. linear\_reactive\_resistance = [ll\*x0 *for* ll *in* line\_lengths] *# реактивное сопротивление ветвей, Ом* 375. starting\_point = [0 *for* i *in* range(32)] *# номер узла, где ветвь начинается* 376. end\_point = [i *for* i *in* range(1, 33)] *# номер узла, где ветвь заканчивается* 378. *# исходные данные узлов: нагрезка <0; источник >0* 379. power = [30, 9, 2, 3, 10, 0.7, 0.8, 0.5, 0.9, 1, 0.9, 0.5, 1, 1.3, 1, 0.8, 0.6, 0.8, 1, 1, 0.7, 1, 0.8, 1, 380. 1.1, 1, 2.2, 1.4, 1.2, 1.3, 1.4, 22] *# нагрузки узлов* 382. sn\_summer = [90, 80, 100, 100, 100] 383. *for* i *in* range(26): 384. sn\_summer.append(50) 385. sn\_summer.append(70) 387. sn\_winter = [95, 90] 388. *for* i *in* range(29): 389. sn\_winter.append(100) 390. sn\_winter.append(55) 392. *if* season == "summer": 393. sn = sn\_summer 394. *else*: 395. sn = sn\_winter 397. s = [] *# полная мощность узлов, МВА* 398. *for* i, pow *in* enumerate(power): 399. p = pow \* sn[i] / 100 400. q = p \* math.tan(math.acos(0.97)) 401. s.append(-complex(p, q)) 403. number\_of\_nodes = len(s)+1 *# количество узлов + 1 базисный* 404. number\_of\_branches = len(starting\_point) *# количество ветвей* 405. base\_node\_number = node\_type.index('base') *# номер базисного узла* 407. u=[base\_node\_voltage+0\*1j *for* i *in* range(number\_of\_nodes-1)] *# начальные приближения напряжений* 409. *# составление первой матрицы инциденций* 410. M\_sum = [[0 *for* j *in* range(number\_of\_branches)] *for* i *in* range(number\_of\_nodes)] 411. *for* i *in* range(number\_of\_branches): 412. M\_sum[starting\_point[i]][i] = 1 413. M\_sum[end\_point[i]][i] = -1 415. *# составление матрицы М, не содержащей базисного угла (удалена 0 строка)* 416. M = M\_sum 417. M = numpy.delete(M, base\_node\_number, axis=0) 419. *# формирование матрицы сопротивлений ветвей (диагональная)* 420. z\_branches = [[0 *for* j *in* range(number\_of\_branches)] *for* i *in* range(number\_of\_branches)] 421. *for* i *in* range(number\_of\_branches): 422. z\_branches[i][i]=linear\_active\_resistance[i]+linear\_reactive\_resistance[i]\*1j 424. *# формирование матрицы узловых проводимостей* 425. y\_nodes = M.dot(numpy.linalg.inv(z\_branches)).dot(M.transpose()) 427. *# матрица сопротивлений узлов* 428. z\_nodes = numpy.linalg.inv(y\_nodes) 430. *# формирование матрицы узловых токов* 431. node\_current = [] 432. *for* i *in* range(number\_of\_nodes-1): 433. node\_current.append(s[i].conjugate()/rated\_voltage.conjugate()) 435. *# расчет напряжений в узлах* 436. result\_u = u+z\_nodes.dot(node\_current) 437. result\_u = numpy.insert(result\_u, base\_node\_number, base\_node\_voltage) 438. result\_abs\_u = abs(result\_u) 440. *# расчет токов в линиях* 441. result\_i = numpy.linalg.inv(z\_branches).dot((numpy.transpose(M\_sum)).dot(result\_u))/3\*\*0.5 442. abs\_result\_i = abs(result\_i) 444. *# расчет потерь в линиях* 445. dp = [] 446. *for* i *in* range(len(linear\_active\_resistance)): 447. I = abs\_result\_i[i] 448. dp.append((I)\*\*2\*3\*linear\_active\_resistance[i]) 450. *return* sum(dp) 451. def *gtu\_initialization*(): 452. *global* m\_to\_hours, m\_kr\_hours 453. current\_datetime = current\_date *# текущая дата* 454. custom\_datetime\_1 = 6 455. custom\_datetime\_2 = 10 456. current\_season = None 457. *if* 6<= current\_datetime.month <10: 458. current\_season = 'summer' 459. *else*: 460. current\_season = 'winter' 461. *# определение мощности потребления* 462. p\_sum = None 463. *if* current\_season == 'summer': 464. p\_sum = 77.6 465. *else*: 466. p\_sum = 89.7 467. p\_sum += get\_power\_loss(10, current\_season) 468. n\_gtu = math.ceil(p\_sum / 16) *# определение числа необходимых ГТУ* 469. actual\_loading = round(p\_sum / n\_gtu / 16, 2) *# определение фактической загрузки ГТУ* 470. hp = hourly\_production(actual\_loading) *# определение отношения мч/ч* 471. engine\_hpd = hp \* 24 *# наработка за сутки при данной нагрузке* 472. *# отключение ГТУ по состоянию* 473. *for* gtu *in* gtes: 474. *if* gtu.state == 1: 475. *if* gtu.kr-engine\_hpd-2.5 <= 0: 476. gtu.stop\_kr() 477. *# плюс к затратам за КР* 478. *elif* gtu.to-engine\_hpd-2.5 <= 0: 479. gtu.stop\_to() 480. *# плюс к затратам за ТО* 481. *else*: 482. *continue* 483. *# отключение ГТУ по количеству n\_gtu* 484. gtes.sort(key=lambda x: x.kr) 485. n = 0 486. *for* gtu *in* gtes: 487. *if* gtu.state == 0 and n < n\_gtu: 488. *if* n < n\_gtu: 489. n += 1 490. *else*: 491. gtu.stop\_n() 493. *# запуск ГТУ по количеству n\_gtu* 494. n = 0 495. *for* gtu *in* gtes: 496. *if* gtu.state == 0: 497. *if* n < n\_gtu: 498. gtu.start\_n() 499. n += 1 500. *# обновление состояний ГТУ по прошествии дня* 501. *for* gtu *in* gtes: 502. *if* gtu.state == 1: 503. gtu.to -= engine\_hpd 504. gtu.kr -= engine\_hpd 505. *elif* gtu.state == 2: 506. gtu.service\_time -= 1 507. *if* gtu.service\_time == 0: 508. gtu.state = 0 509. *else*: 510. *continue* 511. *for* gtu *in* gtes: 512. print(gtu) 513. GTU\_info(gtu.n + 1, gtu.power, gtu.load, f'{gtu.to:.3f}', f'{gtu.kr:.3f}', gtu.state) 514. *###################################################################################* 515. price = None 516. root.bind('<F11>', fullscreen) 517. *# Создаем холст (Canvas) для рисования* 518. canvas = Canvas(root, bg='black', highlightthickness=0) 519. canvas.place(x=0, y=0, width=w, height=h)  *# Растягиваем на весь экран* 520. *# Установки ГТУ* 521. marginG = w \* 0.02  *# Отступы* 522. sizeG = w \* 0.09  *# Размер* 523. num\_gtu = 9 *# Кол-во ГТУ* 524. widthG = num\_gtu \* sizeG + (num\_gtu - 1) \* marginG *#Расчитываем ширину* 525. xG = (w - widthG) / 2 526. yG = w \* 0.015       *# Начальная координата Y* 527. gtu\_on = load\_scaled\_image("img/GTU\_on.png", sizeG) 528. gtu\_off = load\_scaled\_image("img/GTU\_off.png", sizeG) 529. gtu\_repair = load\_scaled\_image("img/GTU\_repair.png", sizeG) 531. *# Создаем 9 изображений ГТУ* 532. GTU\_dict = { 533. 1: {'id': canvas.create\_image(xG, yG, image=gtu\_on, anchor="nw"), 534. 'coords': (xG, yG)}, 535. 2: {'id': canvas.create\_image(xG + (sizeG + marginG)\*1, yG, image=gtu\_on, anchor="nw"), 536. 'coords': (xG + (sizeG + marginG)\*1, yG)}, 537. 3: {'id': canvas.create\_image(xG + (sizeG + marginG)\*2, yG, image=gtu\_on, anchor="nw"), 538. 'coords': (xG + (sizeG + marginG)\*2, yG)}, 539. 4: {'id': canvas.create\_image(xG + (sizeG + marginG)\*3, yG, image=gtu\_on, anchor="nw"), 540. 'coords': (xG + (sizeG + marginG)\*3, yG)}, 541. 5: {'id': canvas.create\_image(xG + (sizeG + marginG)\*4, yG, image=gtu\_on, anchor="nw"), 542. 'coords': (xG + (sizeG + marginG)\*4, yG)}, 543. 6: {'id': canvas.create\_image(xG + (sizeG + marginG)\*5, yG, image=gtu\_on, anchor="nw"), 544. 'coords': (xG + (sizeG + marginG)\*5, yG)}, 545. 7: {'id': canvas.create\_image(xG + (sizeG + marginG)\*6, yG, image=gtu\_on, anchor="nw"), 546. 'coords': (xG + (sizeG + marginG)\*6, yG)}, 547. 8: {'id': canvas.create\_image(xG + (sizeG + marginG)\*7, yG, image=gtu\_on, anchor="nw"), 548. 'coords': (xG + (sizeG + marginG)\*7, yG)}, 549. 9: {'id': canvas.create\_image(xG + (sizeG + marginG)\*8, yG, image=gtu\_on, anchor="nw"), 550. 'coords': (xG + (sizeG + marginG)\*8, yG)} 551. } 552. *# Худ для ГТУ* 553. colorG = "#f0f0f0" *#Настройка цвета* 554. widthG = 3 *#Настройка толщины обводки* 555. shiftG = w \* 0.1 *#Настройка сдвига худа (по умолчанию находится на месте самого ГТУ)* 556. GTU\_huds = { 557. 1: { 558. 'rect': canvas.create\_rectangle(xG, yG + shiftG, xG + sizeG, yG + shiftG + sizeG, outline=colorG, width=widthG), 559. 'coords': (xG, yG + shiftG) 560. }, 561. 2: { 562. 'rect': canvas.create\_rectangle(xG + (sizeG + marginG)\*1, yG + shiftG, xG + sizeG + (sizeG + marginG)\*1, yG + shiftG + sizeG, outline=colorG, width=widthG), 563. 'coords': (xG + (sizeG + marginG)\*1, yG + shiftG) 564. }, 565. 3: { 566. 'rect': canvas.create\_rectangle(xG + (sizeG + marginG)\*2, yG + shiftG, xG + sizeG + (sizeG + marginG)\*2, yG + shiftG + sizeG, outline=colorG, width=widthG), 567. 'coords': (xG + (sizeG + marginG)\*2, yG + shiftG) 568. }, 569. 4: { 570. 'rect': canvas.create\_rectangle(xG + (sizeG + marginG)\*3, yG + shiftG, xG + sizeG + (sizeG + marginG)\*3, yG + shiftG + sizeG, outline=colorG, width=widthG), 571. 'coords': (xG + (sizeG + marginG)\*3, yG + shiftG) 572. }, 573. 5: { 574. 'rect': canvas.create\_rectangle(xG + (sizeG + marginG)\*4, yG + shiftG, xG + sizeG + (sizeG + marginG)\*4, yG + shiftG + sizeG, outline=colorG, width=widthG), 575. 'coords': (xG + (sizeG + marginG)\*4, yG + shiftG) 576. }, 577. 6: { 578. 'rect': canvas.create\_rectangle(xG + (sizeG + marginG)\*5, yG + shiftG, xG + sizeG + (sizeG + marginG)\*5, yG + shiftG + sizeG, outline=colorG, width=widthG), 579. 'coords': (xG + (sizeG + marginG)\*5, yG + shiftG) 580. }, 581. 7: { 582. 'rect': canvas.create\_rectangle(xG + (sizeG + marginG)\*6, yG + shiftG, xG + sizeG + (sizeG + marginG)\*6, yG + shiftG + sizeG, outline=colorG, width=widthG), 583. 'coords': (xG + (sizeG + marginG)\*6, yG + shiftG) 584. }, 585. 8: { 586. 'rect': canvas.create\_rectangle(xG + (sizeG + marginG)\*7, yG + shiftG, xG + sizeG + (sizeG + marginG)\*7, yG + shiftG + sizeG, outline=colorG, width=widthG), 587. 'coords': (xG + (sizeG + marginG)\*7, yG + shiftG) 588. }, 589. 9: { 590. 'rect': canvas.create\_rectangle(xG + (sizeG + marginG)\*8, yG + shiftG, xG + sizeG + (sizeG + marginG)\*8, yG + shiftG + sizeG, outline=colorG, width=widthG), 591. 'coords': (xG + (sizeG + marginG)\*8, yG + shiftG) 592. } 593. } 594. *# Функция для отображения текста в нужных ГТУ* 595. def *GTU\_info*(num, wt, prcnt, hTO, hKR, state): 597. canvas.delete(f"gtu\_{num}")  *# Общий тег для всех элементов* 599. text\_x, text\_y = GTU\_huds[num]["coords"] 600. text\_x += (w \* 0.005) 601. text\_y += (w \* 0.003) 603. img\_x, img\_y = GTU\_dict[num]["coords"]  606. *if* state == 1: 607. canvas.create\_image(img\_x, img\_y, image=gtu\_on, anchor="nw", tags=f"gtu\_{num}") 608. *elif* state == 0: 609. canvas.create\_image(img\_x, img\_y, image=gtu\_off, anchor="nw", tags=f"gtu\_{num}") 610. *elif* state == 2: 611. canvas.create\_image(img\_x, img\_y, image=gtu\_repair, anchor="nw", tags=f"gtu\_{num}") 613. lines = [ 614. f"Номер ГТУ: {num}", 615. f"Номинальная W: {wt}Мвт", 616. f"Уровень загрузки: {prcnt}%", 617. f"Моточасы до ТО: {hTO}ч", 618. f"Моточасы до КР: {hKR}ч", 619. ] 621. *for* i, line *in* enumerate(lines): 622. canvas.create\_text( 623. text\_x, 624. text\_y + i \* int(w \* 0.015), 625. text=line, 626. anchor="nw", 627. fill="white", 628. font=("Arial", int(h\*0.01)), 629. tags=f"gtu\_{num}" 630. ) 631. *# Котлы* 632. marginB = w \* 0.02  *# Отступы* 633. sizeB = w \* 0.09  *# Размер* 634. num\_blr = 6 *# Кол-во котлов* 635. widthB = num\_blr \* sizeB + (num\_blr - 1) \* marginB *#Расчитываем ширину* 636. xB = xG      *# Начальная координата X* 637. yB = w \* 0.26     *# Начальная координата Y* 638. boiler\_on = load\_scaled\_image("img/Boiler\_on.png", sizeG) 639. boiler\_off = load\_scaled\_image("img/Boiler\_off.png", sizeG) 640. *# Создаем 6 изображений Котлов* 641. BLR\_dict = { 642. 1: { 643. 'id': canvas.create\_image(xB, yB, image=boiler\_on, anchor="nw"), 644. 'coords': (xB, yB) 645. }, 646. 2: { 647. 'id': canvas.create\_image(xB + (sizeB + marginB)\*1, yB, image=boiler\_on, anchor="nw"), 648. 'coords': (xB + (sizeB + marginB)\*1, yB) 649. }, 650. 3: { 651. 'id': canvas.create\_image(xB + (sizeB + marginB)\*2, yB, image=boiler\_on, anchor="nw"), 652. 'coords': (xB + (sizeB + marginB)\*2, yB) 653. }, 654. 4: { 655. 'id': canvas.create\_image(xB + (sizeB + marginB)\*3, yB, image=boiler\_on, anchor="nw"), 656. 'coords': (xB + (sizeB + marginB)\*3, yB) 657. }, 658. 5: { 659. 'id': canvas.create\_image(xB + (sizeB + marginB)\*4, yB, image=boiler\_on, anchor="nw"), 660. 'coords': (xB + (sizeB + marginB)\*4, yB) 661. }, 662. 6: { 663. 'id': canvas.create\_image(xB + (sizeB + marginB)\*5, yB, image=boiler\_on, anchor="nw"), 664. 'coords': (xB + (sizeB + marginB)\*5, yB) 665. } 666. } 667. *# Худ для Котлов* 668. colorB = "#f0f0f0" *#Настройка цвета* 669. widthB = 3 *#Настройка толщины обводки* 670. shiftB = w \* 0.1 *#Настройка сдвига худа (по умолчанию находится на месте самого ГТУ)* 671. BLR\_huds = { 672. 1: { 673. 'rect': canvas.create\_rectangle(xB, yB + shiftB, xB + sizeB, yB + shiftB + sizeB, outline=colorB, width=widthB), 674. 'coords': (xB, yB + shiftB) 675. }, 676. 2: { 677. 'rect': canvas.create\_rectangle(xB + (sizeB + marginB)\*1, yB + shiftB, xB + sizeB + (sizeB + marginB)\*1, yB + shiftB + sizeB, outline=colorB, width=widthB), 678. 'coords': (xB + (sizeB + marginB)\*1, yB + shiftB) 679. }, 680. 3: { 681. 'rect': canvas.create\_rectangle(xB + (sizeB + marginB)\*2, yB + shiftB, xB + sizeB + (sizeB + marginB)\*2, yB + shiftB + sizeB, outline=colorB, width=widthB), 682. 'coords': (xB + (sizeB + marginB)\*2, yB + shiftB) 683. }, 684. 4: { 685. 'rect': canvas.create\_rectangle(xB + (sizeB + marginB)\*3, yB + shiftB, xB + sizeB + (sizeB + marginB)\*3, yB + shiftB + sizeB, outline=colorB, width=widthB), 686. 'coords': (xB + (sizeB + marginB)\*3, yB + shiftB) 687. }, 688. 5: { 689. 'rect': canvas.create\_rectangle(xB + (sizeB + marginB)\*4, yB + shiftB, xB + sizeB + (sizeB + marginB)\*4, yB + shiftB + sizeB, outline=colorB, width=widthB), 690. 'coords': (xB + (sizeB + marginB)\*4, yB + shiftB) 691. }, 692. 6: { 693. 'rect': canvas.create\_rectangle(xB + (sizeB + marginB)\*5, yB + shiftB, xB + sizeB + (sizeB + marginB)\*5, yB + shiftB + sizeB, outline=colorB, width=widthB), 694. 'coords': (xB + (sizeB + marginB)\*5, yB + shiftB) 695. } 696. } 697. *# Функция для отображения текста в нужных ГТУ* 698. def *BLR\_info*(num, prcnt, pwr, state): 699. canvas.delete(f"blr\_{num}") 700. img\_x, img\_y = BLR\_dict[num]["coords"] 702. *# Координаты для текста* 703. text\_x, text\_y = BLR\_huds[num]["coords"] 704. text\_x += (w \* 0.005) 705. text\_y += (w \* 0.003) 707. *if* state:  *# True или "on"* 708. canvas.create\_image(img\_x, img\_y, image=boiler\_on, anchor="nw", tags=f"blr\_{num}") 709. *else*:  *# False или "off"* 710. canvas.create\_image(img\_x, img\_y, image=boiler\_off, anchor="nw", tags=f"blr\_{num}") 711. lines = [ 712. f"Номер котла: {num}", 713. f"Уровень загрузки: {prcnt}%", 714. f"Мощность: {pwr}Гкал" 715. ] 717. *for* i, line *in* enumerate(lines): 718. canvas.create\_text( 719. text\_x, 720. text\_y + i \* int(w \* 0.015), 721. text=line, 722. anchor="nw", 723. fill="white", 724. font=("Arial", int(h\*0.01)), 725. tags=f"blr\_{num}"  *# Тот же тег* 726. ) 727. *# Константы для кнопок управления датой* 728. BUTTON\_WIDTH = 17  *# Ширина в символах* 729. BUTTON\_HEIGHT = 1   *# Высота в линиях текста* 730. BUTTON\_FONT = ('Arial', 10)  *# Шрифт для кнопок* 731. *# Создаем фрейм для группировки кнопок управления датой* 732. date\_control\_frame = Frame(root, bg='black') 733. date\_control\_frame.place(relx=0.02, rely=0.95, anchor=SW)  *# Фиксируем в нижнем левом углу* 734. *# Создаем кнопки внутри фрейма* 735. previous\_month\_button = Button(date\_control\_frame, text='Предыдущий месяц', 736. command=previous\_month, 737. width=BUTTON\_WIDTH, 738. height=BUTTON\_HEIGHT, 739. font=BUTTON\_FONT) 740. previous\_month\_button.pack(side=LEFT, padx=5, pady=2) 741. previous\_date\_button = Button(date\_control\_frame, text='Предыдущий день', 742. command=previous\_date, 743. width=BUTTON\_WIDTH, 744. height=BUTTON\_HEIGHT, 745. font=BUTTON\_FONT) 746. previous\_date\_button.pack(side=LEFT, padx=5, pady=2) 747. next\_date\_button = Button(date\_control\_frame, text='Следующий день', 748. command=next\_date, 749. width=BUTTON\_WIDTH, 750. height=BUTTON\_HEIGHT, 751. font=BUTTON\_FONT) 752. next\_date\_button.pack(side=LEFT, padx=5, pady=2) 753. next\_month\_button = Button(date\_control\_frame, text='Следующий месяц', 754. command=next\_month, 755. width=BUTTON\_WIDTH, 756. height=BUTTON\_HEIGHT, 757. font=BUTTON\_FONT) 758. next\_month\_button.pack(side=LEFT, padx=5, pady=2) 759. *# Метка с датой* 760. date\_label = Label(date\_control\_frame, 761. text=current\_date.strftime("%d.%m.%Y"), 762. fg='white', 763. bg='black', 764. font=('Arial', 14, 'bold')) 765. date\_label.pack(side=LEFT, padx=10) 766. *# Кнопка загрузки (оставляем в правом нижнем углу)* 767. download\_button = Button(root, 768. text='Загрузить данные', 769. bg='white', 770. command=open\_excel\_files, 771. font=BUTTON\_FONT) 772. download\_button.place(relx=0.98, rely=0.95, anchor=SE) 773. *# Параметры прямоугольника (добавьте в начало с другими параметрами)* 774. rect\_x1, rect\_y1 = BLR\_dict[6]["coords"] 775. rect\_y1 += (h \* 0.037) 776. rect\_x1 = xG + (sizeG + marginG)\*6 777. rect\_x2 = xG + sizeG + (sizeG + marginG)\*8 778. rect\_y2 = yB + shiftB + sizeB 779. rect\_outline\_color = "#f0f0f0"  *# цвет обводки* 780. rect\_outline\_width = 3  *# Толщина обводки* 781. rect\_font = ("Arial", 20)  *# Шрифт* 782. *# Создаем прямоугольник с разными координатами* 783. status\_rect = canvas.create\_rectangle( 784. rect\_x1, rect\_y1, 785. rect\_x2, rect\_y2, 786. outline=rect\_outline\_color, 787. width=rect\_outline\_width, 788. ) 789. text\_padding\_x = (w \* 0.005)  *# Горизонтальный отступ* 790. text\_padding\_y = (w \* 0.004)  *# Вертикальный отступ* 791. *# Создаем текстовый элемент в левом верхнем углу прямоугольника* 792. status\_text = canvas.create\_text( 793. rect\_x1 + text\_padding\_x,  *# X: левый край + отступ* 794. rect\_y1 + text\_padding\_y,  *# Y: верхний край + отступ* 795. text="СТАТУС СИСТЕМЫ\n\nОжидание данных...", 796. fill="white", 797. font=rect\_font, 798. justify=LEFT,  *# Выравнивание по левому краю* 799. width=(rect\_x2 - rect\_x1) - 2\*text\_padding\_x,  *# Ширина с учетом отступов* 800. anchor="nw"  *# Привязка к северо-западному углу (левому верхнему)* 801. ) 802. *# Функция для обновления текста (оставляем без изменений)* 803. def *set\_status\_message*(message): 804. """Устанавливает текст сообщения в прямоугольнике""" 805. canvas.itemconfig(status\_text, text=message) 806. canvas.update\_idletasks() 807. *# Примеры использования:* 808. *# set\_status\_message("ВНИМАНИЕ!\n\nОбнаружена ошибка\nКод: 45")* 809. *# set\_status\_message("РАБОТА ЗАВЕРШЕНА\n\nВсе процессы\nостановлены")* 811. root.mainloop() |