ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ГОРОДА МОСКВЫ

Государственное бюджетное общеобразовательное учреждение города Москвы

«Школа № 1387»

(ГБОУ школа № 1387)

**Кейс №5: Сбор и обработка данных температуры**

Авторы работы:

Марков Александр Сергеевич

Прошина Евгения Александровна

Аваков Артём Артурович

11 И класс

ГБОУ школа № 1387

Научный руководитель:

Москва, 2020

Оглавление

[Анализ технических требований 2](#_Toc34672010)

[Задание 2](#_Toc34672011)

[Условия выполнения 2](#_Toc34672012)

[Пользовательский интерфейс 3](#_Toc34672013)

[Язык программирования и программные средства 3](#_Toc34672014)

[**Языки программирования** 3](#_Toc34672015)

[Сравнение языков программирования 4](#_Toc34672016)

[Модули стандартной библиотеки Python 4](#_Toc34672017)

[Библиотеки Python 4](#_Toc34672018)

[**Программные средства** 4](#_Toc34672019)

[Описание основных этапов разработки 4](#_Toc34672020)

[Первый этап 4](#_Toc34672021)

[Второй этап 5](#_Toc34672022)

[Третий этап 5](#_Toc34672023)

[Четвёртый этап 5](#_Toc34672024)

[Пятый этап 5](#_Toc34672025)

[Шестой этап 5](#_Toc34672026)

[Структурная и функциональная схема 6](#_Toc34672027)

[Функциональные схемы: Parser 6](#_Toc34672028)

[Алгоритм работы программного продукта 7](#_Toc34672029)

[База данных 10](#_Toc34672030)

[Результаты разработки 10](#_Toc34672031)

[Программный код 11](#_Toc34672032)

[Модуль 1 11](#_Toc34672033)

[База данных 15](#_Toc34672034)

[Таймер 18](#_Toc34672035)

# **Анализ технических требований**

## **Задание**

Реализовать программный модуль для сбора, хранения и обработки данных с удалённых температур данных

## **Условия выполнения**

Посредством специализированного сервиса, расположенного по адресу <http://dt.miet.ru/ppo_it>, осуществить сбор данных об уличной температуре в 16 городах. Необходимо использовать показатели датчиков, находящихся в 10 квартирах не менее, чем пяти районов города. Время осуществления – 48 часов реального времени.

Обращение к сервису происходит не реже, чем один раз в 10 минут.

Полученные данные хранятся с помощью реляционной СУБД, реализованной на основе ER-модели.

Взаимодействие с данными осуществляется через пользовательский интерфейс, который по запросу пользователя отображает информацию о температуре в квартирах и на улице в виде графика/диаграммы (см. пункт [пользовательский интерфейс](#_Пользовательский_интерфейс))

Пользовательский интерфейс является кроссплатформенным, а также организован в соответствии со стандартами построения UI

Для облегчения работы с вносящимися в программный код изменениями используется система управления версиями git

## **Пользовательский интерфейс**

Согласно регламенту испытаний функционал UI подразумевает выведение следующих данных в виде графика/диаграммы/величины:

* Данные температуры в реальном времени в определённой квартире
* График изменения уличной температуре на протяжении суток реального времени в одном из городов
* График изменения средней температуры в квартирах в одном из городов на протяжении суток реального времени
* График изменения температуры в одной квартире в каждом из городов
* Диаграмма максимальных температур в квартирах в каждом из районов (не менее пяти) одного города

# **Язык программирования и программные средства**

## **Языки программирования**

Учитывая изложенные выше особенности технического задания, было решено, что в качестве основного языка программирования будет использоваться Python.

При выборе важную роль сыграли такие его качества, как минималистичный синтаксиса ядра и богатая стандартная библиотека, позволяющая работать с высокоуровневыми структурами данных и взаимодействовать со многими сетевыми протоколами, в частности HTTP.

Управление БД осуществляется посредством СУБД SQLite ввиду того, что его легко использовать при кроссплатформенном переносе, а также он очень надёжен с точки зрения программного кода.

### Сравнение языков программирования

|  |  |  |  |
| --- | --- | --- | --- |
|  | Python | C++ | Java |
| Опыт работы | + | - | - |
| Простота синтаксиса и удобство | + | - | - |
| Производительность | - | + | + |
| Работа с БД | + | - | - |
| Динамическая типизация | + | - | - |

*Таблица 1*

Производительность языка не играла важную роль при выборе языка, так как программный продукт используется малым числом лиц и объёмы обрабатываемых данных относительно малы

### Модули стандартной библиотеки Python

* модуль requests – инструмент составления HTTP и GET-запросов для взаимодействия с сервером
* модуль time – для работы с реальным/серверным временем
* модуль sqlite3 – кроссплатформенное средство работы с БД

### Библиотеки Python

* Matplotlib – осуществляет визуализацию информации из БД в UI
* IPyWidgets – интерактивные HTML виджеты для Jupyter Notebook

## **Программные средства**

Дистрибутив Anaconda предустанавливает инструмент интерактивной разработки Jupyter Notebook, удобный с точки зрения разработки и использования интерфейса, в том числе он поддерживает создание графического интерфейса для пользователя. В качестве IDE используется Spyder из дистрибутива Anaconda. Одна из важных особенностей этой среды разработки – это интеграция с научными библиотеками Python, к примеру Matplotlib

# **Описание основных этапов разработки**

## **Первый этап**

С помощью интернет-ресурсов мы изучили синтаксис и принципы работы описанных выше модулей и библиотек языка Python, СУБД SQLite, а также ознакомились с основами работы в программных средствах Jupyter Notebook и Spyder

## **Второй этап**

Для реализации программного кода выбраны методы структурного программирования. Составлены концепции функций, образующих модуль взаимодействия с сервером и обработкой полученной информации и модуль пользовательского интерфейса и визуализации данных (далее «модуль 1» и «модуль 2» соответственно; см. [Алгоритм работы программного продукта](#_Алгоритм_работы_программного))

## **Третий этап**

На сайте <https://github.com/> создана система контроля версий продукта, доступная по следующей ссылке: <https://github.com/CangCiwei/PredProf>. Составлен план выполнения технического задания согласно функциональности продукта.

## **Четвёртый этап**

Проектируется UI. Осуществляется написание программного кода для модуля 1 и модуля 2. Код комментируется. Каждая составная функцию тестируется на корректность возвращаемых данных.

Оформляется техническая документация.

## **Пятый этап**

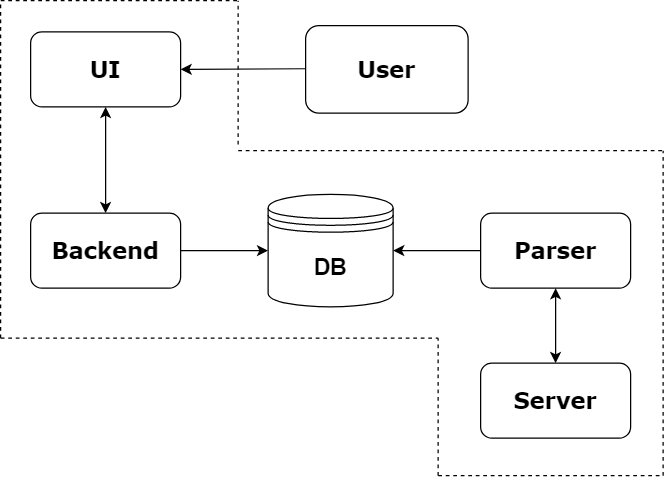
Тестируется функциональность программы. Проводятся необходимые корректировки программного кода (см. [Третий этап](#_Третий_этап)) и повторные тестирования.

## **Шестой этап**

Производится сбор данных температуры на протяжении 48 часов реального времени. По ним строятся графики, описанные в разделе [Пользовательский интерфейс](#_Пользовательский_интерфейс)

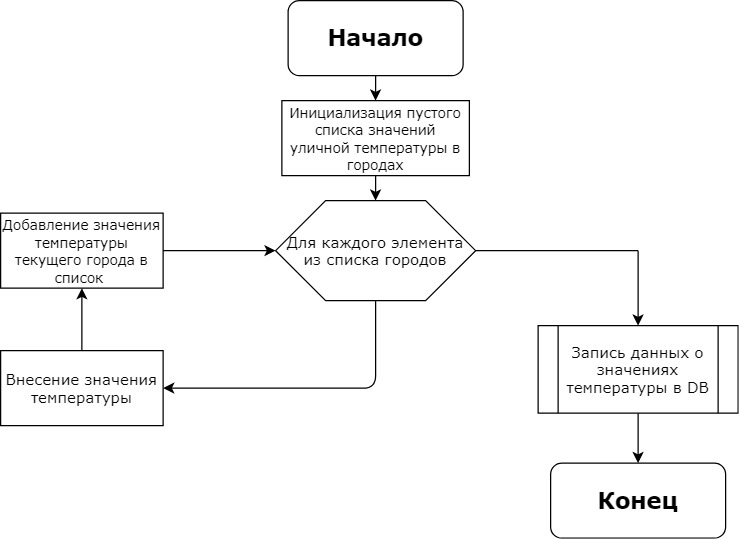
# Структурная и функциональная схема

1. Структурная схема работы алгоритма

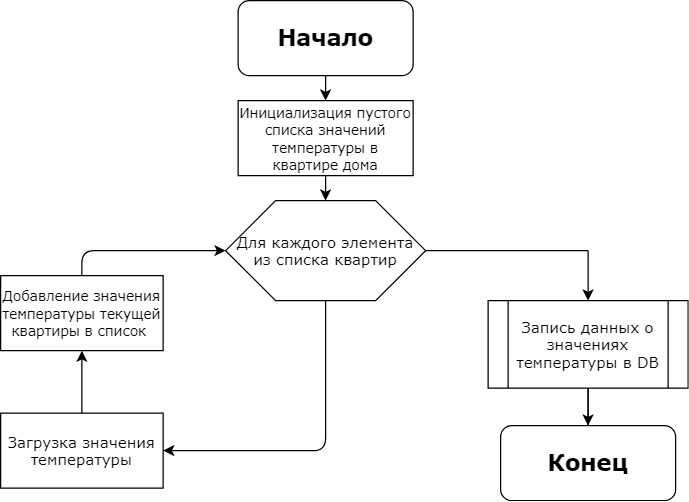


## Функциональные схемы: Parser

1. Функциональная схема: внесения данных уличной температуры в городах



1. Функциональная схема: внесение значений температуры в квартире



# Алгоритм работы программного продукта

Пользователь посредством специализированного UI взаимодействует с данными находящимися в DB посредством Backend

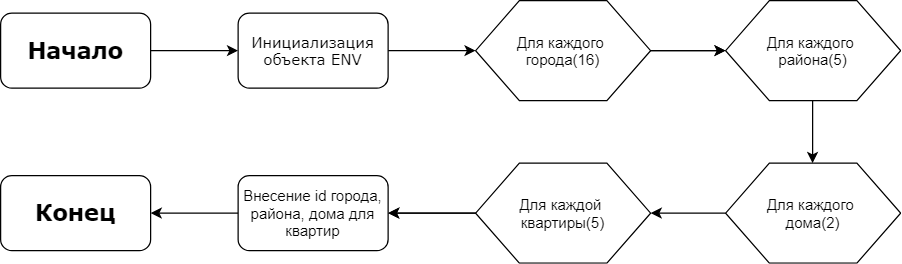
Раздел Backend производит визуальное отображение информации в качестве графиков и диаграмм, которое реализовано с помощью Matplotlib, также здесь осуществлена некоторая обработка информации в соответствии с регламентом, например, определение максимальных значений температуры.

DB хранит информацию, полученную с сервера в виде нескольких таблиц: таблица показаний уличной температуры в городах, таблица показаний температуры в множестве квартир одного города, таблица показаний температуры в одной квартире в каждом из городов, таблица показаний температуры в квартирах в нескольких (5) районах одного города.

Parser производит обработку данных полученных сервиса и передаёт их в DB в соответствующие таблицы. В том числе, с помощью модуля requests здесь извлекается токен для взаимодействия с сервиса, производится извлечение данных на разных уровнях сервера; создаются «цели» для создания графика «температур множества квартир» и здесь же находится подпрограмма, отвечающая за время сбора данных с сервиса, реализованная с помощью модуля time

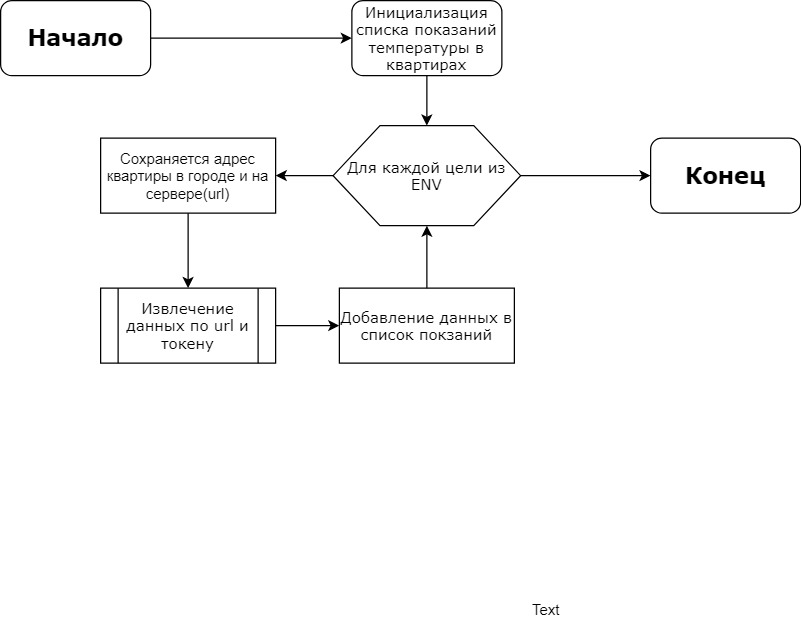
Ниже представлены функциональные схемы работы некоторых частей алгоритма:

1. Загрузка «целей» - адресов квартир, из которых будут считываться данные температуры



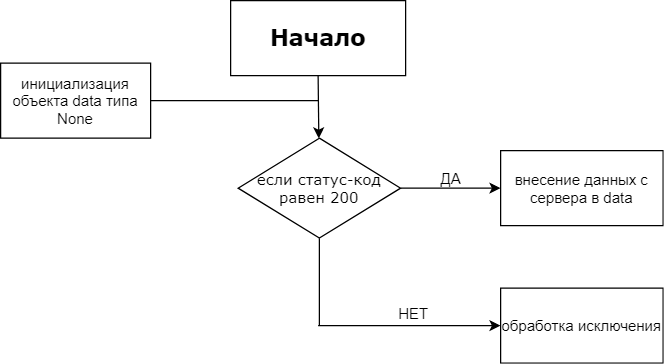
ENV – это объект типа dict, содержащий список целей targets, url-адрес сервиса, token и время работы программы.

1. Сбор данных о каждой цели



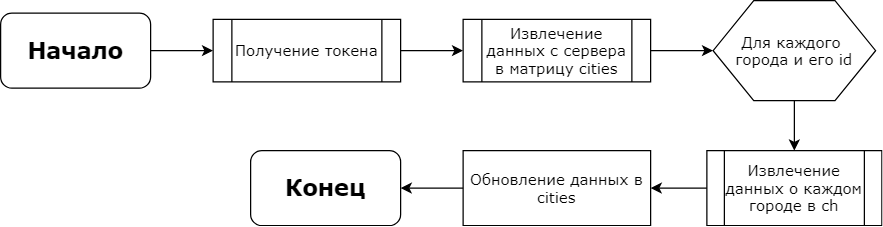
Алгоритм возвращает список показаний температуры в квартирах

1. Извлечение данных с сервера



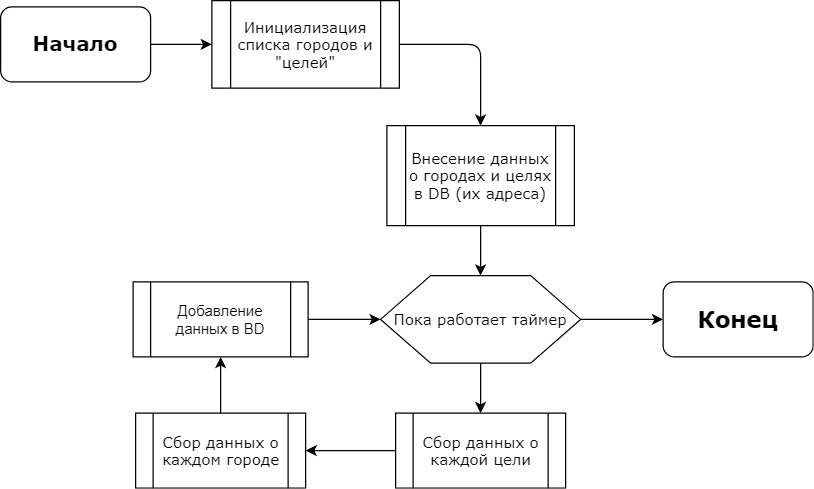
Алгоритм возвращает data в формате json

1. Извлечение подробных данных о каждом городе



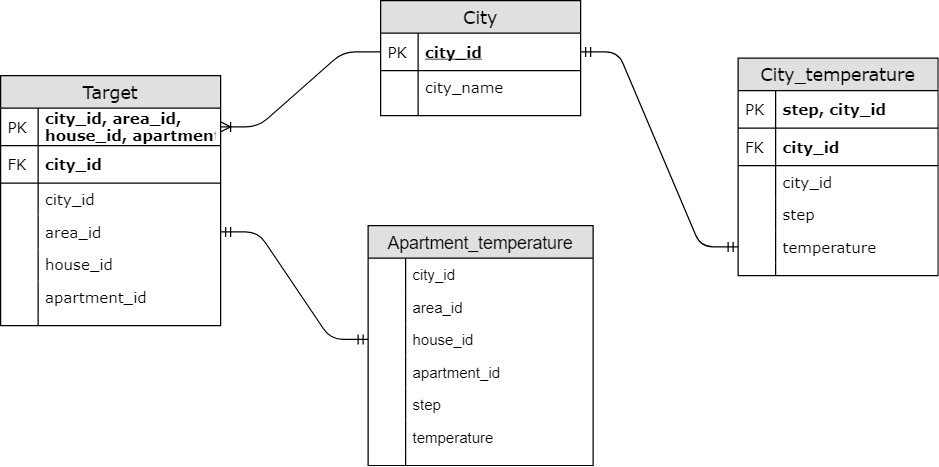
Дополнительно, перед использованием объектов cities и ch они проверяются на корректность (не тип None). Алгоритм возвращает cities – объект типа dict. Использование этой программы подразумевает постоянные изменения в показаниях температуры

1. Основной алгоритм



## **База данных**

1. ER-модель



# Результаты разработки

Реализован программный продукт, позволяющий выводить данные температуры в различных квартирах и регионах. Разработан программный модуль, обеспечивающий взаимодействие с сервисом. Полученная с него информация хранится в реляционной базе данных. Взаимодействие с пользователем происходит через специализированный UI. Интерфейс является самоадаптирующимся для различных OC/имеет единый стиль, общий для всех OC/гибридное решение.

Выполнены все пункты технического задания, описанные в начале документации.

# Программный код

## **Модуль 1**

1. *#! /usr/bin/env python3*
2. u"""Модуль сбора данных."""
3. **import** os
4. **import** json
5. **import requests**
6. **from** stimer **import** STimer
7. **from** random **import** shuffle
8. **from** database **import** DB
10. **ENV = {**
11. "token\_path": "../token",
12. "cache": ".cache.json",
13. "targets": [],
14. "url": "http://dt.miet.ru/ppo\_it/api",
15. **"dbname": "../database.db",**
16. "delay": 60,
17. "end": 24 \* 3600
18. }
20. **with open(ENV["token\_path"]) as file:**
21. ENV["token"] = file.read().strip()

24. **def** init\_random(size, max\_val):
25. **"""Генерация диапазона случайных значений."""**
26. keys = list(range(1, max\_val + 1))
27. shuffle(keys)
28. **return** dict.fromkeys(keys[0:size])

31. **def** gen\_random\_targets(cities):
32. """Генерация целей для сбора данных."""
33. targets = {}
34. **for** city **in** cities:
35. ***# случайные номера значения районов***
36. areas = init\_random(4, city["area\_count"])
37. **for** i, area\_id **in** enumerate(areas.keys()):
38. *# список для хранения номеров домов для текущего города и района*
39. *# запрос данных для*
40. **res = load\_data(f"{ENV['url']}/{city['city\_id']}/{area\_id}")**
41. houses = init\_random(2, len(res["data"]["data"]))
42. **for** j, house\_id **in** enumerate(houses.keys()):
43. url = f"{ENV['url']}/{city['city\_id']}/{area\_id}/{house\_id}"
44. res2 = load\_data(url)
45. **size = 1**
46. **if** i == 3 **and** j == 1:
47. size += 2
48. apartments = init\_random(
49. size, res2["data"]["data"]["apartment\_count"]
50. **)**
51. houses[house\_id] = list(apartments.keys())
52. areas[area\_id] = houses
53. targets[city["city\_id"]] = areas
54. **return** targets

57. **def** make\_targets(cities):
58. """Создание списка целей."""
59. targets\_map = gen\_random\_targets(cities)
60. **targets = []**
61. **for** city\_id **in** targets\_map.keys():
62. target = {}
63. target["city\_id"] = city\_id
64. **for** area\_id **in** targets\_map[city\_id]:
65. **target["area\_id"] = area\_id**
66. **for** house\_id **in** targets\_map[city\_id][area\_id]:
67. target["house\_id"] = house\_id
68. **for** apartment\_id **in** targets\_map[city\_id][area\_id][house\_id]:
69. target["apartment\_id"] = apartment\_id
70. **targets.append(target.copy())**
71. **return** targets

74. **def** load\_data(url=ENV["url"], token=ENV["token"]):
75. **"""Функция загрузки данных."""**
76. result = {"data": None, "error": None}
77. **try**:
78. **print**(f">>> REQ: {url} ")
79. res = requests.get(url, headers={"X-Auth-Token": token})
80. **print(f"<<< RES: {res.status\_code}\n\t{res.text}")**
81. **if** res.status\_code == 200:
82. result["data"] = res.json()
83. **else**:
84. result["error"] = res.text
85. **except Exception as error:**
86. **print**(error)
87. **return** result

90. **def get\_cities():**
91. res = load\_data()
92. cities = []
93. **if** res["data"] **is** **not** None:
94. cities = res["data"]["data"]
95. **return cities**

98. **def** get\_cities\_data(cities=None):
99. """ извлечение подробных данных о каждом городе """
100. **if cities is None:**
101. cities = get\_cities()
102. **for** city **in** cities:
103. res = load\_data(f"{ENV['url']}/{city['city\_id']}")
104. **if** res["data"] **is** **not** None:
105. **city.update(res["data"]["data"])**
106. **return** cities

109. **def** get\_apatments\_data(targets):
110. **"""сбор данных о каждой цели"""**
111. data\_set = []
112. **for** target **in** targets:
113. url = f"{ENV['url']}/{target['city\_id']}/{target['area\_id']}/{target['house\_id']}/{target['apartment\_id']}"
114. res = load\_data(url, ENV["token"])
115. **dataset = {**
116. "city\_id": target['city\_id'],
117. "area\_id": target['area\_id'],
118. "house\_id": target['house\_id'],
119. "apartment\_id": target['apartment\_id'],
120. **"temperature": None**
121. }
122. **if** res["data"] **is** **not** None:
123. dataset["temperature"] = res["data"]["data"]["temperature"]
124. **else**:
125. **dataset["temperature"] = -273**
126. data\_set.append(dataset)
127. **return** data\_set

130. **def get\_one\_target(city, area, house, apartment):**
131. """Сбор данных об одной цели в реальном времени"""
132. data = load\_data(f"{ENV['url']}/{city}/{area}/{house}/{apartment}")
133. **return** data
135. **def read\_cache():**
136. """Чтение закэшированных значений городов и целей"""
137. data = None
138. **if** os.path.isfile(ENV["cache"]):
139. **with** open(ENV["cache"]) **as** fd:
140. **data = json.load(fd)**
141. **return** data
143. **def** write\_cache():
144. **with** open(ENV["cache"], "w") **as** fd:
145. **json.dump({"cities": ENV["cities"], "targets": ENV["targets"]}, fd)**
147. **def** initialize():
148. """Функция инициализации."""
149. data = read\_cache()
150. **if data is not None:**
151. ENV["cities"] = data["cities"]
152. ENV["targets"] = data["targets"]
153. **else**:
154. *# получение списка городов*
155. **ENV["cities"] = get\_cities()**
156. *# генерация списка целей*
157. ENV["targets"] = make\_targets(get\_cities\_data(ENV["cities"]))
158. write\_cache()

161. **def** main():
162. """Главная функция."""
163. initialize()
164. **if** os.path.isfile(ENV["dbname"]):
165. **os.remove(ENV["dbname"])**
166. db = DB(ENV["dbname"])
167. db.add\_cities(ENV["cities"])
168. db.add\_targets(ENV["targets"])
169. timer = STimer(ENV["end"])
170. **count = 0**
171. **while** **not** timer.is\_stop():
172. *# Главный цикл сбора данных*
173. cities\_data = get\_cities\_data(ENV["cities"])
174. targets\_data = get\_apatments\_data(ENV["targets"])
175. **print(f"Received {len(cities\_data) + len(targets\_data)} out of {len(ENV['cities'])+len(ENV['targets'])} objects")**
176. **print**("Saving...")
177. **for** city **in** get\_cities\_data(ENV["cities"]):
178. db.add\_city\_temperature(
179. city\_id=city["city\_id"],
180. **step=count,**
181. temperature=city["temperature"]
182. )
183. **for** target **in** get\_apatments\_data(ENV["targets"]):
184. db.add\_apartment\_temperature(
185. **city\_id=target["city\_id"],**
186. area\_id=target["area\_id"],
187. house\_id=target["house\_id"],
188. apartment\_id=target["apartment\_id"],
189. step=count,
190. **temperature=target["temperature"]**
191. )
192. count += 1
193. **print**(f"Timeout {ENV['delay']}s.")
194. timer.sleep(ENV['delay'])
196. **if** \_\_name\_\_ == "\_\_main\_\_":
197. main()

## **База данных**

1. *#! /usr/bin/env python3*
2. """Модуль работы с БД sqlite3."""
3. **import** sqlite3
4. **import** sys

7. **class** DB:
8. """Класс-обертка над sqlite3."""
10. **def \_\_init\_\_(self, name: str):**
11. """Метод инициализации.
12. name - имя БД
13. """
14. *# открытие соединения с БД*
15. **self.db\_connection = sqlite3.connect(name)**
16. *# создание таблиц в БД*
17. self.\_create\_tables()
19. **def** \_\_del\_\_(self):
20. **"""Destructor."""**
21. *# закрытие соединения с БД*
22. self.db\_connection.close()

25. **def add\_city\_temperature(self, \*, city\_id, step, temperature):**
26. """Метод добавляет запись в таблицу city\_temperature.
27. city\_id - идентификатор (номер) города
28. step - шаг измерений
29. temperature - значение температуры
30. **"""**
31. **try**:
32. self.db\_connection.executescript(
33. "**\n**".join([
34. "INSERT INTO city\_temperature VALUES (",
35. **f"{city\_id},",**
36. f"{step},",
37. f"{temperature}",
38. f");"
39. ])
40. **)**
41. **except** Exception **as** err:
42. *# вывод сообщения об ошибке в стандартный поток ошибок*
43. **print**(err)
45. **def add\_apartment\_temperature(self, \*, city\_id, area\_id, house\_id, apartment\_id, step, temperature):**
46. """Метод добавляет запись в таблицу apartment\_temperature.
47. city\_id - идентификатор (номер) города
48. area\_id - идентификатор (номер) района
49. house\_id - идентификатор (номер) дома
50. **step - шаг измерений**
51. temperature - значение температуры
52. """
53. **try**:
54. self.db\_connection.executescript(
55. **"\n".join([**
56. "INSERT INTO apartment\_temperature VALUES (",
57. f"{city\_id},",
58. f"{area\_id},",
59. f"{house\_id},",
60. **f"{apartment\_id},",**
61. f"{step},",
62. f"{temperature}",
63. f");"
64. ])
65. **)**
66. **except** Exception **as** err:
67. *# вывод сообщения об ошибке в стандартный поток ошибок*
68. sys.stderr.write(err)
70. **def get\_apartment\_temperature(self, \*, city\_id, area\_id, house\_id, apartment\_id):**
71. """"""
72. **return** self.db\_connection.executescript(
73. "**\n**".join([
74. "SELECT \* FROM apartment\_temperature WHERE",
75. **f"city\_id = {city\_id} and",**
76. f"area\_id = {area\_id} and",
77. f"house\_id = {house\_id} and",
78. f"apartment\_id = {apartment\_id};"
79. ])
80. **)**
82. **def** add\_targets(self, targets):
83. """Добавление целей для сбора данных."""
84. sql = "INSERT INTO target VALUES"
85. **values = ""**
86. **for** target **in** targets:
87. sql += f"**\n\t**({target['city\_id']}, {target['area\_id']}, {target['house\_id']}, {target['apartment\_id']}),"
88. sql = f"{sql[0:-1]};"
89. self.db\_connection.executescript(sql)
91. **def** add\_cities(self, cities):
92. """Добавление городов в БД"""
93. sql = "".join([
94. "INSERT INTO city VALUES**\n\t**",
95. **',\n\t'.join([f'({city["city\_id"]}, \'{city["city\_name"]}\')' for city in cities]),**
96. ";"
97. ])
98. self.db\_connection.executescript(sql)
100. **def \_create\_tables(self):**
101. """Создание таблицы БД"""
102. SQL = """PRAGMA FOREIGN\_KEYS = on;
103. CREATE TABLE IF NOT EXISTS city(
104. city\_id int PRIMARY KEY NOT NULL,
105. **city\_name text NOT NULL**
106. );
108. CREATE TABLE IF NOT EXISTS city\_temperature(
109. city\_id int NOT NULL,
110. **step int NOT NULL,**
111. temperature int NOT NULL,
112. CONSTRAINT pk\_city\_temperature PRIMARY KEY (step, city\_id),
113. FOREIGN KEY (city\_id) REFERENCES city(city\_id)
114. );
115. **CREATE INDEX IF NOT EXISTS idx\_city\_tmpr\_cid ON city\_temperature(city\_id);**
117. CREATE TABLE IF NOT EXISTS target(
118. city\_id int NOT NULL,
119. area\_id int NOT NULL,
120. **house\_id int NOT NULL,**
121. apartment\_id int NOT NULL,
122. CONSTRAINT pk\_target PRIMARY KEY (city\_id, area\_id, house\_id, apartment\_id),
123. FOREIGN KEY (city\_id) REFERENCES city(city\_id)
124. );
125. **CREATE INDEX IF NOT EXISTS idx\_target\_cid ON target(city\_id);**
126. CREATE INDEX IF NOT EXISTS idx\_target\_cid\_aid ON target(city\_id, area\_id);
127. CREATE INDEX IF NOT EXISTS idx\_target\_cid\_aid\_hid ON target(city\_id, area\_id, house\_id);
129. CREATE TABLE IF NOT EXISTS apartment\_temperature(
130. **city\_id int NOT NULL,**
131. area\_id int NOT NULL,
132. house\_id int NOT NULL,
133. apartment\_id int,
134. step int,
135. **temperature int**
136. );
137. CREATE INDEX IF NOT EXISTS idx\_aprtm\_tmpr ON apartment\_temperature(city\_id, area\_id, house\_id, apartment\_id);
138. """
139. self.db\_connection.executescript(SQL)

## **Таймер**

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 |  | *#! /usr/bin/env .python3*  *"""The Simple timer."""*  **import** time  **class** STimer**:**  *"""The simple timer class."""*  **def** \_\_init\_\_**(**self**,** duration**):**  *"""Initializer."""*  self**.**start **=** self**.**timestamp**()**  self**.**duration **=** duration  **@classmethod**  **def** timestamp**(**cls**):**  *"""Current time stamp."""*  **return** int**(**time**.**time**())**  **def** is\_stop**(**self**):**  *"""Check."""*  **return** self**.**timestamp**()** **-** self**.**start **>=** self**.**duration  **def** sleep**(**self**,** nsecond**=0):**  *"""Sleep."""*  time**.**sleep**(**nsecond**)** |