Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра интеллектуальных информационных технологий

**Лабораторная работа №3 по курсу «ПБЗ» на тему:**

**«**Приложение для работы с онтологий**»**

Выполнила студент

группы 921731:

Проверил: .

**МИНСК**

2022

**СОДЕРЖАНИЕ**

1. Введение
2. Обоснование выбора языка и средств реализации
3. Структура онтологии <ученые>
4. Описание разработанного приложения
5. Вывод
6. Список используемых источников

В данной лабораторной работе нам предлагалось написать собственное приложение для работы с онтологиями. Онтология - это формализованная область знаний с помощью концептуальной схемы, такая схема, как правило, состоит из структуры данных, содержащей все релевантные классы объектов, их связи и правила, которые были приняты в данной области. Онтология используется в процессе программирования как форма представления знаний о реальном мире и его части.

Современные онтологии строятся по большей части одинаково, независимо от языка написания. Обычно они состоят из [экземпляров](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D0%B7%D0%B5%D0%BC%D0%BF%D0%BB%D1%8F%D1%80_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0), [понятий](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BD%D1%8F%D1%82%D0%B8%D0%B5), [атрибутов](https://ru.wikipedia.org/w/index.php?title=%D0%90%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82_(%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C)&action=edit&redlink=1) и [отношений](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_(%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0)&action=edit&redlink=1):

*Экземпляры* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *instances*) или индивиды ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *individuals*) — это объекты, основные нижнеуровневые компоненты онтологии; могут представлять собой как физические объекты (люди, дома, планеты), так и [абстрактные](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%86%D0%B8%D1%8F) (числа, слова).

*Понятия* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *concepts*) или *классы* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *classes*) — абстрактные группы, коллекции или наборы объектов. Они могут включать в себя экземпляры, другие классы либо же сочетания и того, и другого.

Объекты в онтологии могут иметь *атрибуты*. Каждый атрибут имеет по крайней мере имя и значение и используется для хранения информации, которая специфична для объекта и привязана к нему.

Важная роль атрибутов заключается в том, чтобы определять *отношения* (зависимости) между объектами онтологии. Обычно отношением является атрибут, значением которого является другой объект.

В последние годы увеличивается необходимость создания онтологий, это связано в первую очередь с тем, что это необходимо для совместного использования людьми или программными агентами общего понимания структуры информации, для возможности повторного использования знаний в предметной области, для отделения знаний в предметной области от оперативных знаний, для анализа знаний в предметной области и др.

**Protégé**  это [свободный](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5" \o "Свободное программное обеспечение), [открытый](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5" \o "Открытое программное обеспечение) редактор [онтологий](https://ru.wikipedia.org/wiki/%D0%9E%D0%BD%D1%82%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Онтология (информатика)) и [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA" \o "Фреймворк) для построения [баз знаний](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B7%D0%BD%D0%B0%D0%BD%D0%B8%D0%B9" \o "База знаний).

Платформа Protégé поддерживает два основных способа моделирования онтологий посредством редакторов Protégé-Frames и Protégé-OWL. Онтологии, построенные в Protégé, могут быть экспортированы во множество форматов, включая [RDF](https://ru.wikipedia.org/wiki/RDF" \o "RDF) ([RDF Schema](https://ru.wikipedia.org/wiki/RDF_Schema" \o "RDF Schema)), [OWL](https://ru.wikipedia.org/wiki/Web_Ontology_Language" \o "Web Ontology Language) и [XML Schema](https://ru.wikipedia.org/wiki/XML_Schema" \o "XML Schema).

Protégé имеет открытую, легко расширяемую архитектуру за счёт поддержки модулей расширения функциональности.

Protégé поддерживается значительным сообществом, состоящим из разработчиков и учёных, правительственных и корпоративных пользователей, использующих его для решения задач, связанных со знаниями, в таких разнообразных областях, как [биомедицина](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%BE%D0%BC%D0%B5%D0%B4%D0%B8%D1%86%D0%B8%D0%BD%D0%B0" \o "Биомедицина), сбор знаний и корпоративное моделирование.

Protégé доступен для свободного скачивания с официального сайта вместе с [плагинами](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B3%D0%B8%D0%BD" \o "Плагин) и онтологиями.

**Онтоло́гия** в информатике ([новолат.](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%D0%BC_(%D0%BB%D0%B5%D0%BA%D1%81%D0%B8%D0%BA%D0%B0)" \l "novolat" \o "Интернационализм (лексика)) *ontologia* от [др.-греч.](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B5%D0%B2%D0%BD%D0%B5%D0%B3%D1%80%D0%B5%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Древнегреческий язык) ὤν род. п. ὄντος — сущее, то, что существует и λόγος — учение, наука) — это попытка всеобъемлющей и подробной [формализации](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F" \o "Формализация) некоторой области [знаний](https://ru.wikipedia.org/wiki/%D0%97%D0%BD%D0%B0%D0%BD%D0%B8%D0%B5" \o "Знание) с помощью [концептуальной схемы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%86%D0%B5%D0%BF%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D1%85%D0%B5%D0%BC%D0%B0" \o "Концептуальная схема). Обычно такая схема состоит из [структуры данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85" \o "Структура данных), содержащей все [релевантные](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D0%B5%D0%B2%D0%B0%D0%BD%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C" \o "Релевантность) классы объектов, их связи и правила ([теоремы](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B5%D0%BC%D0%B0" \o "Теорема), ограничения), принятые в этой области. Этот термин в информатике является производным от древнего философского понятия «[онтология](https://ru.wikipedia.org/wiki/%D0%9E%D0%BD%D1%82%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F" \o "Онтология)».

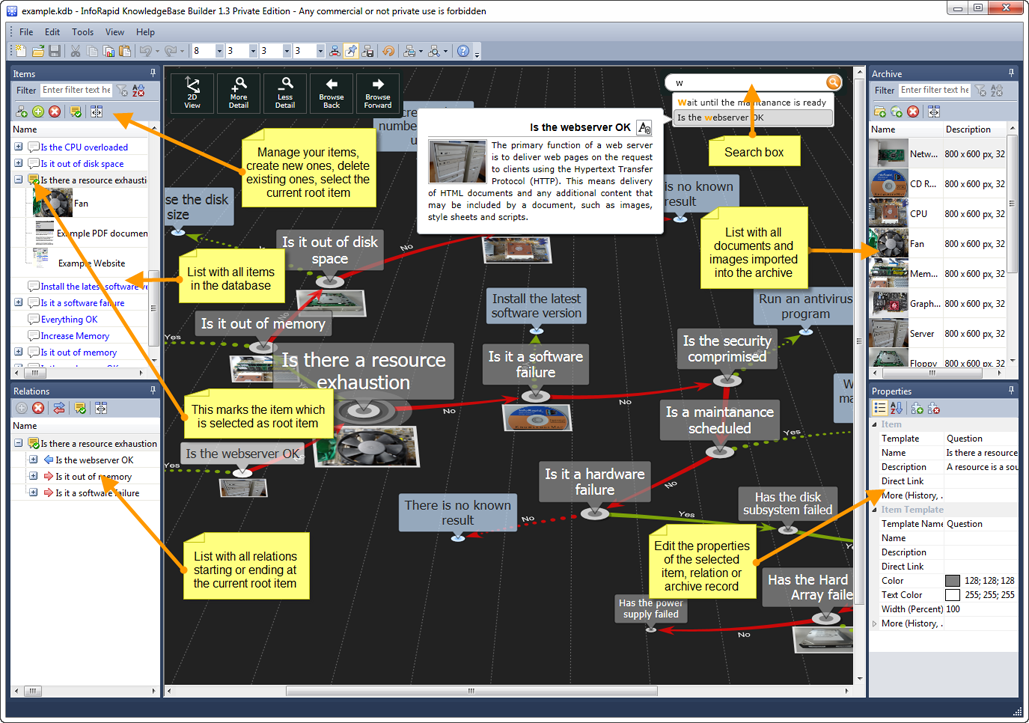
Онтологии используются в [процессе](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F" \o "Процесс разработки программного обеспечения) [программирования](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5" \o "Программирование) как форма [представления знаний](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B7%D0%BD%D0%B0%D0%BD%D0%B8%D0%B9" \o "Представление знаний) о реальном мире или его части. Основные сферы применения — [моделирование бизнес-процессов](https://ru.wikipedia.org/w/index.php?title=%D0%91%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5&action=edit&redlink=1" \o "Бизнес-моделирование (страница отсутствует)), [семантическая паутина](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D0%BD%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BF%D0%B0%D1%83%D1%82%D0%B8%D0%BD%D0%B0" \o "Семантическая паутина) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Английский язык) *Semantic Web*), [искусственный интеллект](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BA%D1%83%D1%81%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D0%BB%D0%BB%D0%B5%D0%BA%D1%82" \o "Искусственный интеллект).

**База знаний** (БЗ; [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Английский язык) *knowledge base, KB*) — [база данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85" \o "База данных), содержащая правила вывода и [информацию](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F" \o "Информация) о человеческом опыте и знаниях в некоторой [предметной области](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D0%BC%D0%B5%D1%82%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D0%BB%D0%B0%D1%81%D1%82%D1%8C" \o "Предметная область) (ISO/IEC/IEEE 24765-2010, ISO/IEC 2382-1:1993). В самообучающихся системах база знаний также содержит информацию, являющуюся результатом решения предыдущих задач.

Современные базы знаний работают совместно с системами [поиска](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0" \o "Поисковая система) и [извлечения](https://ru.wikipedia.org/wiki/%D0%98%D0%B7%D0%B2%D0%BB%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%B8" \o "Извлечение информации) информации. Для этого требуется некоторая модель [классификации понятий](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F" \o "Классификация) и определённый формат [представления знаний](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B7%D0%BD%D0%B0%D0%BD%D0%B8%D0%B9" \o "Представление знаний). Иерархический способ представления в базе знаний набора [понятий](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BD%D1%8F%D1%82%D0%B8%D0%B5" \o "Понятие) и их связей называется [онтологией](https://ru.wikipedia.org/wiki/%D0%9E%D0%BD%D1%82%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Онтология (информатика)).

Онтологию некоторой области знаний вместе со сведениями о свойствах [конкретных объектов](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D0%B7%D0%B5%D0%BC%D0%BF%D0%BB%D1%8F%D1%80_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0" \o "Экземпляр класса) часто называют «базой знаний». Вместе с тем полноценные базы знаний (в отличие от обычной базы данных) содержат в себе не только фактическую информацию, но и [правила вывода](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_%D0%B2%D1%8B%D0%B2%D0%BE%D0%B4%D0%B0" \o "Машина вывода), позволяющие делать [автоматические умозаключения](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%B4%D0%BE%D0%BA%D0%B0%D0%B7%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE_%D1%82%D0%B5%D0%BE%D1%80%D0%B5%D0%BC" \o "Автоматическое доказательство теорем) об уже имеющихся или вновь вводимых фактах и тем самым производить [семантическую](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D0%BD%D1%82%D0%B8%D0%BA%D0%B0" \o "Семантика) (осмысленную) обработку информации.

Область наук об [искусственном интеллекте](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BA%D1%83%D1%81%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D0%BB%D0%BB%D0%B5%D0%BA%D1%82" \o "Искусственный интеллект), изучающая базы знаний и методы работы со знаниями, называется [инженерией знаний](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D0%B8%D1%8F_%D0%B7%D0%BD%D0%B0%D0%BD%D0%B8%D0%B9" \o ")



**ОБОСНОВАНИЕ ВЫБОРА ЯЗЫКА И РЕАЛИЗАЦИИ**

Python — объектно-ориентированный язык с динамической типизацией, указание типа для переменных в нем не требуется. Одной из популярных сфер применения Python является разработка веб-сайтов и веб-приложений, в частности, с помощью фреймворка Django. Другой распространенной сферой является машинное обучение, а также различные программы, связанные с искусственным интеллектом. Но также с помощью Python можно создавать и другие приложения, например, различные десктопные консольные и графические программы и даже мобильные приложения. Нередко данный язык также применяется для написания кастомных сценариев в рамках других программ. Он совместим со многими операционными системами. У Python простой синтаксис, так что его легко читать. Кроме того, в нем нет синтаксических скобок, большого количества модификаторов, C-подобных конструкций и т. д. У него много библиотек с открытым кодом.

Используемые библиотеки

1. Rdflib
2. Tkinter

RDFLib - это библиотека Python для работа с RDF , простым, но мощным языком для представления информации. Благодаря этой библиотеке Python является одним из основных языков манипулирования RDF, другим является Java. Эта библиотека содержит парсеры / сериализаторы почти для всех известных сериализаций RDF, таких как RDF / XML, Turtle, N-Triples и JSON-LD, многие из которых теперь поддерживаются в их обновленной форме (например, Turtle 1.1). Библиотека также содержит как хранимые в памяти, так и постоянные серверные части Graph для хранения информации RDF и многочисленные вспомогательные функции для объявления пространств имен графов, размещения запросов SPARQL.

Tkinter (от англ. tk interface) - это графическая библиотека, позволяющая создавать программы с оконным интерфейсом. Эта библиотека является интерфейсом к популярному языку программирования и инструменту создания графических приложений tcl/tk. Tkinter, как и tcl/tk, является кроссплатформенной библиотекой и может быть использована в большинстве распространённых операционных систем (Windows, Linux, Mac OS X и др.).

Так как Tkinter является достаточно прозрачным интерфейсом к tcl/tk, то основным источником информации для неё являются man-страницы tcl/tk. Эти страницы имеются в любой Unix-системе (в разделе n или 3tk).

**ОБОСНОВАНИЕ ВЫБОРА ЯЗЫКА И РЕАЛИЗАЦИИ**

Для сравнения были выбраны такие языки как Python, C++, C#, Go и Java.

Python — объектно-ориентированный язык с динамической типизацией, указание типа для переменных в нем не требуется. Одной из популярных сфер применения Python является разработка веб-сайтов и веб-приложений, в частности, с помощью фреймворка Django. Другой распространенной сферой является машинное обучение, а также различные программы, связанные с искусственным интеллектом. Но также с помощью Python можно создавать и другие приложения, например, различные десктопные консольные и графические программы и даже мобильные приложения. Нередко данный язык также применяется для написания кастомных сценариев в рамках других программ. Он совместим со многими операционными системами. У Python простой синтаксис, так что его легко читать. Кроме того, в нем нет синтаксических скобок, большого количества модификаторов, C-подобных конструкций и т. д. У него много библиотек с открытым кодом.

C++ — гибридный язык программирования (поддерживает как процедурное, так и объектно-ориентированное программирование). Нередко С++ находит свое применение в системном программировании, в частности, при создании операционных систем, драйверов, различных утилит, антивирусов и т.д. Но только системным программированием применение данного языка не ограничивается. С++ можно использовать в программах любого уровня, где важны скорость работы и производительность. Также он применяется для создания графических приложений, различных прикладных программ, для создания игр с богатой насыщенной визуализацией, используется в мобильном направлении, в веб-разработке для создания веб-приложений или каких-то вспомогательных сервисов, которые обслуживают веб-приложения. В общем С++ - язык широкого пользования, на котором можно создавать практически любые виды программ.

C# — объектно-ориентированный язык программирования. C# является языком с Си-подобным синтаксисом. Язык C# был создан специально для работы с фреймворком .NET. .NET представляет единую для всех поддерживаемых языков библиотеку классов. И какое бы приложение мы не собирались писать на C# так или иначе мы задействуем библиотеку классов .NET.

Go — компилируемый многопоточный язык программирования, разработанный внутри компании Google. Язык Go предназначен для создания различного рода приложений, но прежде всего это веб-сервисы и клиент-серверные приложения. Хотя также язык обладает возможностями по работе с графикой, низкоуровневыми возможностями и т.д.

Java — объектно-ориентированный язык программирования. Ключевой особенностью языка Java является то, что его код сначала транслируется в специальный байт-код, независимый от платформы. А затем этот байт-код выполняется виртуальной машиной JVM (Java Virtual Machine). Подобная архитектура обеспечивает кроссплатформенность и аппаратную переносимость программ на Java, благодаря чему подобные программы без перекомпиляции могут выполняться на различных платформах. Java является языком с Си-подобным синтаксисом.

Рассмотрим парадигмы выбранных языков.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Возможность** | **Языки** | | | | |
| C++ | C# | Go | Java | Python |
| Объектно-ориентированная | + | + | +/− | + | + |
| Функциональная | + | +/− | +/− | −/+ | + |
| Распределенная | +/− | −/+ | + | + | −/+ |

Возьмем такие возможности как, функциональная возможность и распределенная возможность. Функциональная возможность позволяет записывать программу как композицию функций. В чистом функциональном языке нет переменных. Так как функции не имеют побочных эффектов, они могут выполняться в любом порядке. В Python присутствует данная возможность; в C# и Go возможность поддерживается не полностью; в Java и C++ возможность поддерживается ограничено. Язык с распределенной возможностью это язык, содержащий специальные конструкции для поддержки распараллеливания программы на несколько компьютеров. В Java и Go присутствует данная возможность; в C++ возможность поддерживается не полностью; в Python и C# возможность поддерживается ограничено.

Далее рассмотрим возможности типизации.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Возможность** | **Языки** | | | | |
| C++ | C# | Go | Java | Python |
| Статистическая типизация | + | + | + | + | − |
| Динамическая типизация | − | + | − | − | + |
| Явная типизация | + | + | + | + | +/− |
| Неявная типизация | −/+ | −/+ | − | − | + |
| Параметрический полиморфизм | − | + | − | + |  |

В статистической типизации переменные и параметры методов/функций связываются с типом в момент объявления и не могут быть изменены позже. А в динамической они связываются с типами в момент присваивания значения (или передачи параметра в метод/функцию), а не в момент объявления переменной или параметра. Одна и та же переменная в разные моменты может хранить значения разных типов. Только C# обладает двумя выше описанными типизациями; C++, Go и Java используют статистическую типизацию; Python - динамическую. В явной типизации типы переменных и параметров указываются явно, в неявной - не указываются явно. C#, C++, Go и Java используют явную типизацию; в Python она поддерживается не полностью. Python использует неявную типизацию; в C# и C++ она поддерживается ограничено; а в Go и Java вовсе не поддерживается. Параметрический полиморфизм подразумевает возможность указывать constraints или type classes для типов-параметров. Поддерживается в C# и Java, а в C++ и Go не поддерживается.

Далее рассмотрим возможности компиляторов/интерпретаторов.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Возможность** | **Языки** | | | | |
| C++ | C# | Go | Java | Python |
| Многопоточная компиляция | + | − | + | + |  |
| Интерпретатор командной строки | +/− | + | − | − | + |
| Условная компиляция | + | + | +/− | −/+ |  |

Многопоточная компиляция - это возможность компилятора на многопроцессорных системах использовать несколько потоков для ускорения компиляции. Поддерживается в C++, Go и Java; в C# не поддерживается. Интерпретатор командной строки - возможность вводить инструкции языка строка за строкой с их немедленным выполнением. Поддерживается в C# и Python; в Go и Java не поддерживается; в C++ она поддерживается ограничено. Условная компиляция - возможность включать/выключать части кода в зависимости от значения символов условной компиляции (например, с помощью #if … #endif в C++). Поддерживается в C++ и C#; в Go поддерживается не полностью; в Java поддерживается ограничено.

Далее рассмотрим возможности управления памятью.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Возможность** | **Языки** | | | | |
| C++ | C# | Go | Java | Python |
| Объекты на стеке | + | + |  | − | − |
| Ручное управление памятью | + | + |  | − | − |
| Сборка мусора | −/+ | + | + | + | + |

Объекты на стеке позволяют создавать экземпляры объектов не в куче, а на стеке. C++ и C# используют данный тип объектов, Python и Java нет. Ручное управление памятью позволяет явно выделять и освобождать память в куче (например, с помощью операторов new и delete в C++). C++ и C# поддерживают ручное управление памятью, Python и Java нет. Сборка мусора позволяет использовать автоматический процесс сборки мусора (освобождения памяти в куче, занятой неиспользуемыми объектами). C#, Go, Python и Java используют сборку мусора, в C++ поддерживается ограничено.

Далее рассмотрим возможности управления потоком вычислений.

Инструкция goto - безусловный переход на метку. Поддерживается в C++, C# и Go; в Python и Java не поддерживается. Поддержка обработки исключений с помощью try/catch или эквивалентной конструкции. Поддерживается в C++, C#, Python и Java; в Go не поддерживается. Поддержка блока finally при обработке исключений или эквивалентной конструкции. Поддерживается в C#, Python и Java; в C++ не поддерживается. Перезапуски - исключения, не раскручивающие стек вызовов. Возможность из места перехвата исключения вернуться в место установки перезапуска. Не поддерживается в C#, для оставшихся нет информации.

Далее рассмотрим типы и структуры данных.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Возможность** | **Языки** | | | | |
| C++ | C# | Go | Java | Python |
| Кортежи | +/− | + | + | − | + |
| Алгебраические типы данных | − | − | +/− | − |  |
| Многомерные массивы | + | + | + | +/− | +/− |
| Динамические массивы | + | +/− | + | +/− | + |
| Ассоциативные массивы | + | + | + | +/− | + |
| Контроль границ массивов | + | + | + | + | + |
| Списковые включения | − | −/+ |  | − | + |
| Целые числа произвольной длины | − | + | + | + | + |

Возможность вернуть из функции/метода кортеж (tuple) — неименованный тип данных, содержащий несколько безымянных полей произвольного типа. Поддерживается в C#, Go и Python; в Java не поддерживается; в C++ поддерживается не полностью. Алгебраический тип данных — в информатике наиболее общий составной тип, представляющий собой тип-сумму из типов-произведений. Не поддерживается в C++, C# и Java; в Go поддерживается не полностью. Наличие встроенных в язык многомерных массивов. Поддерживается в C++, C# и Go; в Java и Python поддерживается не полностью. Наличие встроенных в язык динамических массивов (способных изменять свой размер во время выполнения программы). Поддерживается в C++ и Go; в Java, C# и Python поддерживается не полностью. Наличие встроенных в язык ассоциативных массивов или хеш-таблиц. Поддерживается в C++, C#, Go и Python; в Java поддерживается не полностью. Наличие списковых включений (или их аналога). Поддерживается в Python; в C++ и Java не поддерживается; в C# поддерживается ограничено. Поддержка целых чисел неограниченной разрядности. Поддерживается в C#, Go, Java и Python; в C++не поддерживается.

Далее рассмотрим объектно-ориентированные возможности.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Возможность** | **Языки** | | | | |
| C++ | C# | Go | Java | Python |
| Интерфейсы | + | + | + | + | + |
| Множественное наследование | + | − | − | − | + |
| Решение конфликта имён при множественном наследовании | + |  |  |  | + |

Интерфейсы - это семантическая и синтаксическая конструкция в коде программы, используемая для специфицирования услуг, предоставляемых классом. Поддерживается во всех рассматриваемых языках. Множественное наследование - это возможность наследовать класс сразу от нескольких классов (не интерфейсов). Поддерживается в C++ и Python; в C#, Go и Java не поддерживается. Решение конфликта имён при множественном наследовании (решение для случая ромбовидного наследования). Решение может приниматься как для всего класса, так и для каждого поля/метода в отдельности. Поддерживается в C++ и Python.

Анонимная функция в программировании — особый вид функций, которые объявляются в месте использования и не получают уникального идентификатора для доступа к ним. Поддерживается в C++, C# и Java; в Python поддерживается не полностью. Наличие в языке макросистемы, обрабатывающей код программы до времени её компиляции и/или выполнения. Поддерживается в C++ и Python; в C# поддерживается не полностью; в Java не поддерживается. Наличие в данном статически типизированном языке инструмента для обобщенного программирования, наподобие шаблонов в C++. Поддерживается в C++, C# и Java; в Go не поддерживается. Возможность перегрузки функций/методов по количеству и типам параметров. Поддерживается в C++, C# и Java; в Go и Python не поддерживается. Возможность создавать переменные, имеющие динамическую область видимости. Поддерживается в C#; в Go, C++, C# и Python не поддерживается. Возможность при вызове функции/метода опускать некоторые параметры, чтобы при этом подставлялось значение по умолчанию, указанное при определении функции. Поддерживается в C++, C#, Go и Python; в Java не поддерживается. Возможность определять локальную функцию внутри другой функции/метода. Подразумевается возможность использовать внутри локальной функции локальные переменные из внешнего блока. Поддерживается в C++, C#, Go и Python; в Java поддерживается не полностью.

Основными причинами для выбора Java являются широкий выбор RDF-хранилищ и наиболее глубокие знания в работе с данным языком программирования.

**СТРУКТУРА ОНТОЛОГИИ <ученые>**

Для погружения в разрабатываемую онтологию была выбрана тема “ученые”, так как для хранения информации о ученых, изобретениях удобен именно иерархический формат хранения данных за счет того, что изоберетения и могут одновременно представляться только в одной выставке.

Для удобства был разработан пользовательский интерфейс, содержащий поля ввода, вывода и кнопки для перехода на соответствующий экран.

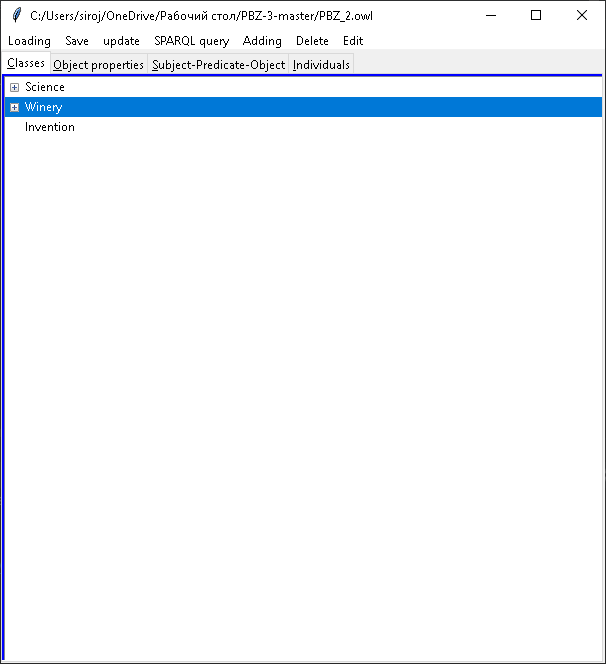


рис. 1. Главное меню пользовательского интерфейса

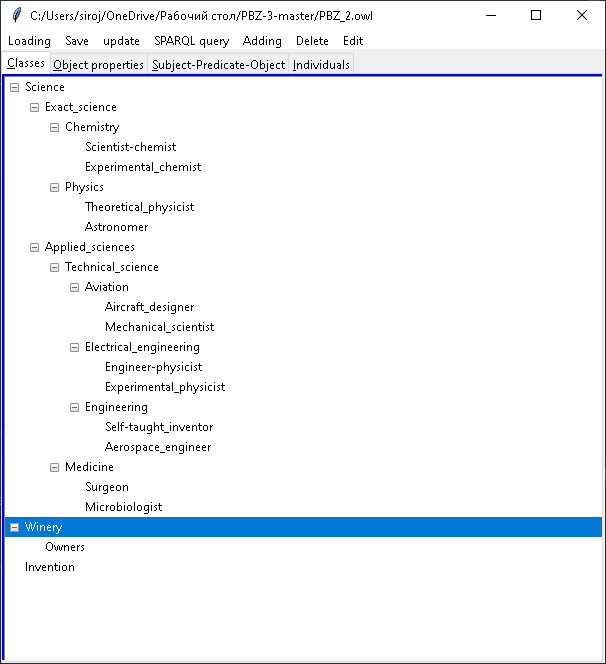


рис. 2. Пользовательский интерфейс после перехода по кнопке “Classes”

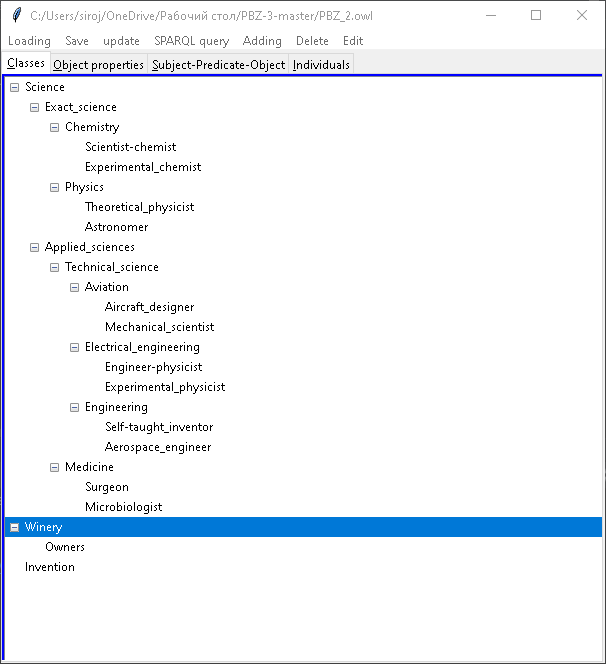


рис. 3. Пользовательский интерфейс после перехода по кнопке “Objects Properties”

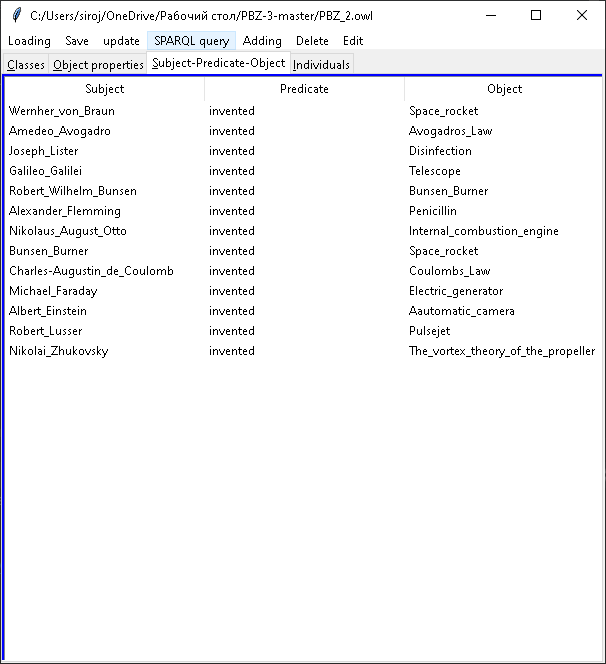


рис. 4. Пользовательский интерфейс после перехода по кнопке “Subject-Predicate-Object”

В разделе “Edit” пользователь имеет право не только добавлять и удалять уже существующие классы, объекты и отношения, но и редактировать уже существующие - для этого есть соответствующие кнопки внизу каждого из экранов.

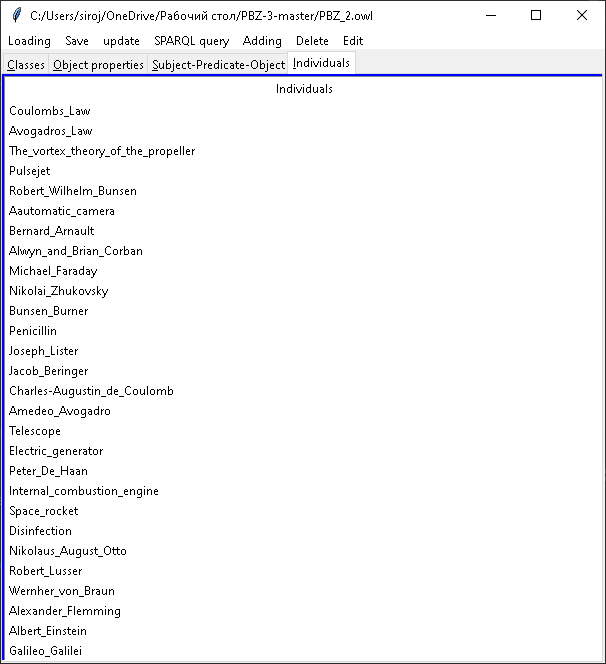


рис. 5. Пользовательский интерфейс после перехода по кнопке “Individual”

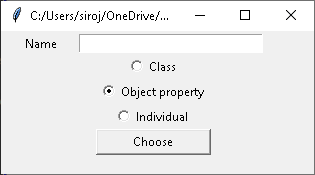


рис. 6. Реализация в пользовательском интерфейсе возможности создание класса или объекта или свойства по одному параметру

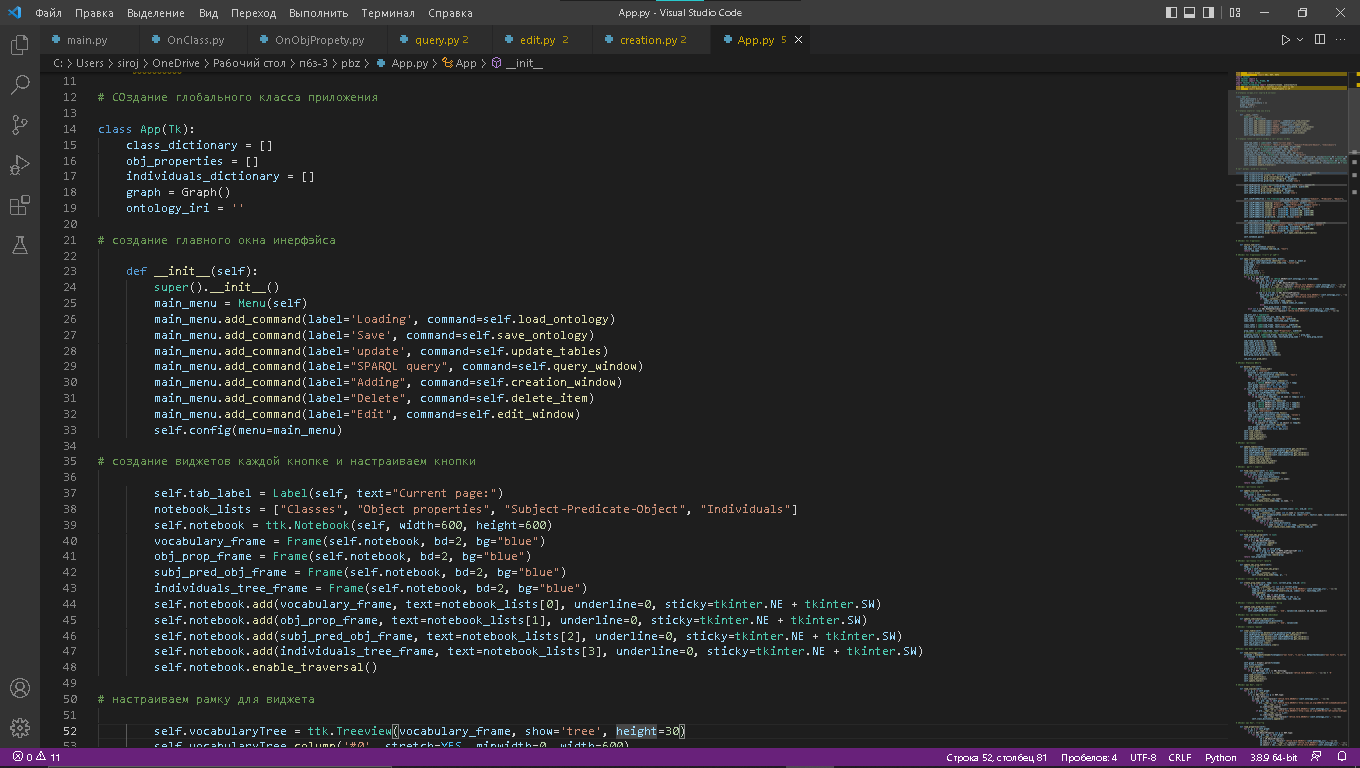


Рис .7

Создание главного окна приложения и его интерфэйс созданная нашей командой при помощи Tkinter и главный мозг нашей онтологии созданной при помощи rdfLib это фрэймвор для создания приложения для работы с онтологией

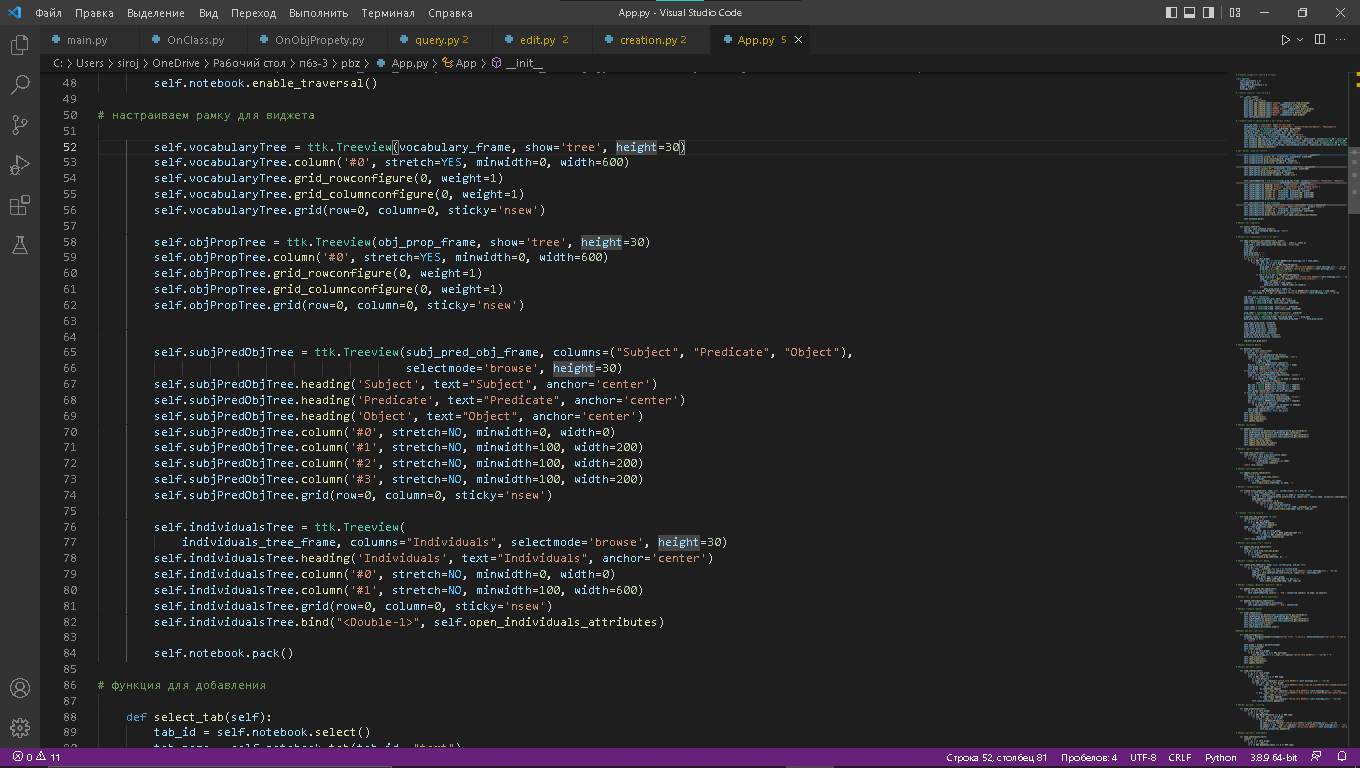


Рис. 8

Настраиваем параметры главного окна для нашего приложения созданной как Frame

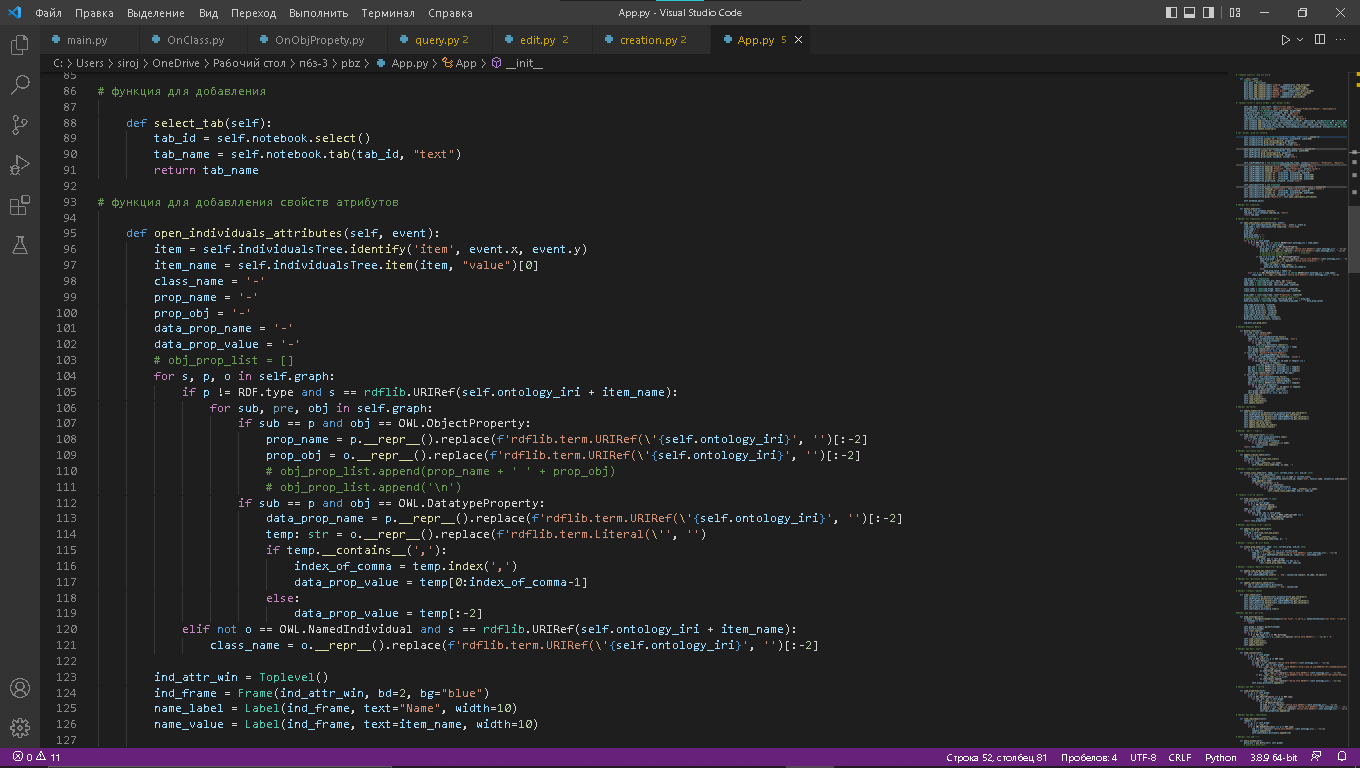


Рис. 9

Создания функции для добавления онтологии в наше приложения и функции для добавления свойств атрибутов в наши свойста

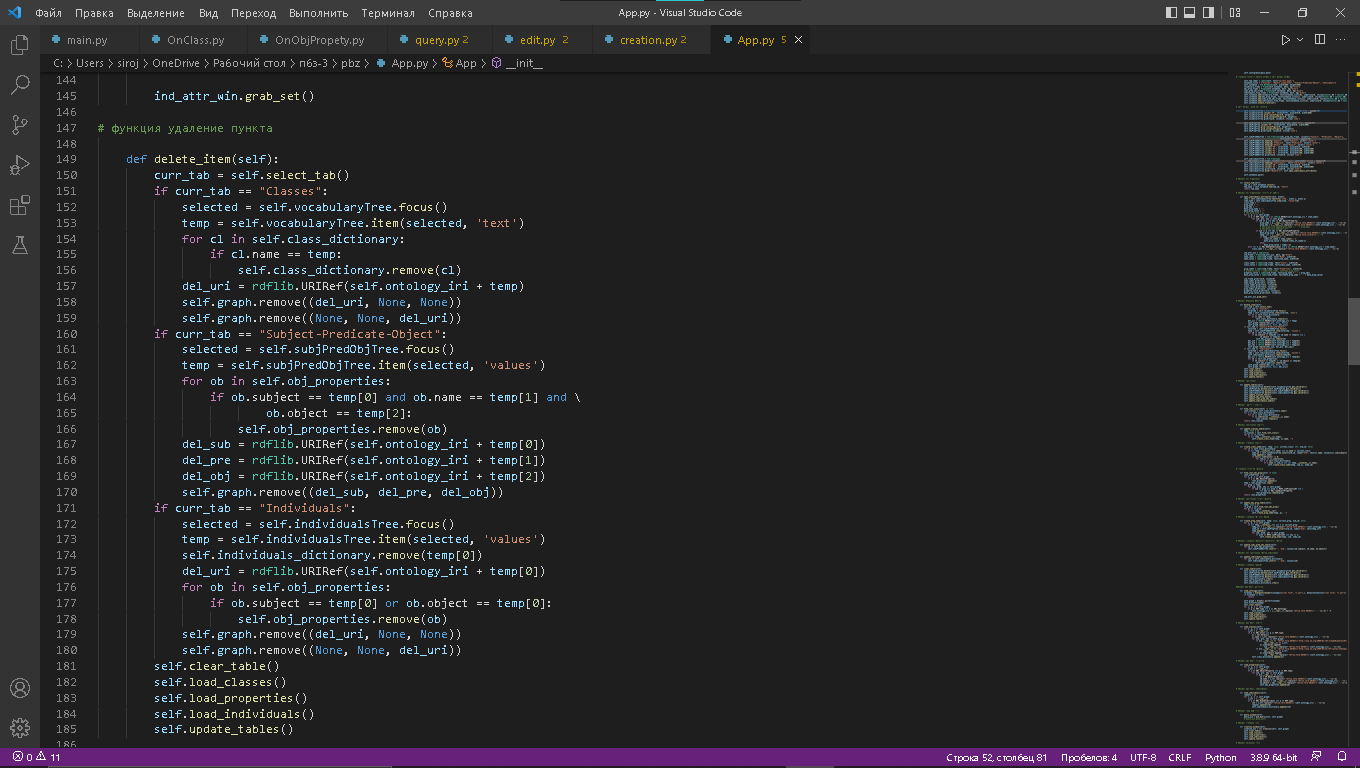


Рис. 10

Функция для удаление выбранного пункта в онтологии

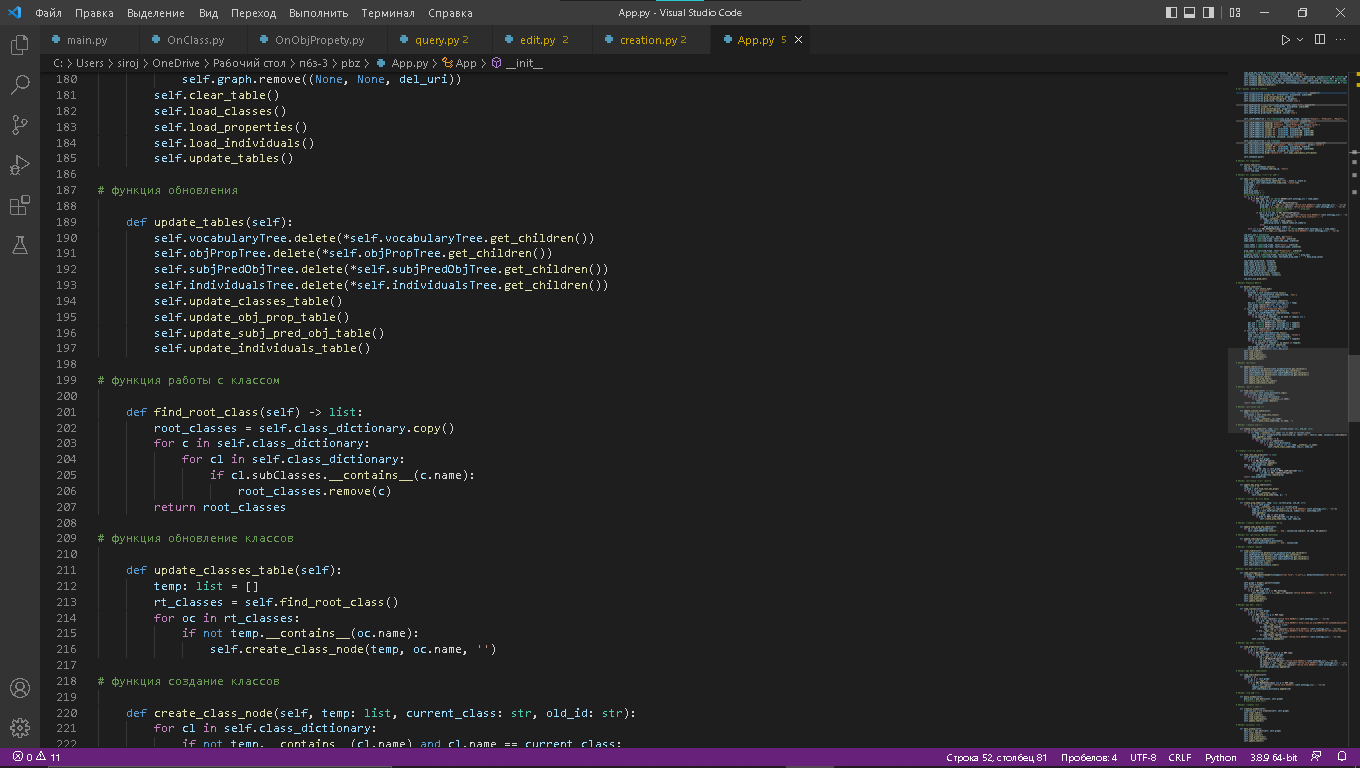


Рис. 11

Создание функции для обновления, для работы с классами и для обновления классов

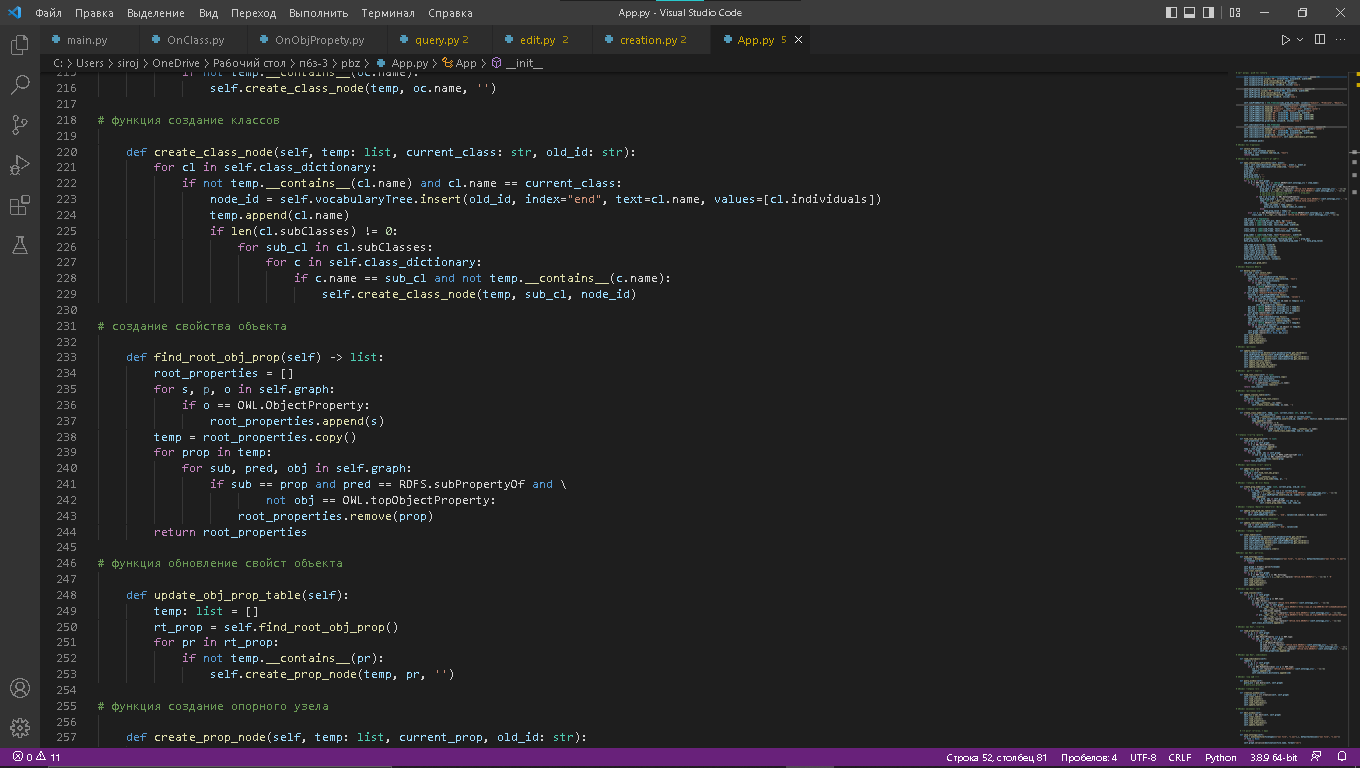


Рис. 12

Теперь перейдем на классы функционал для работы классов : Создание классов,

создание свойств классов и обновление свойств классов

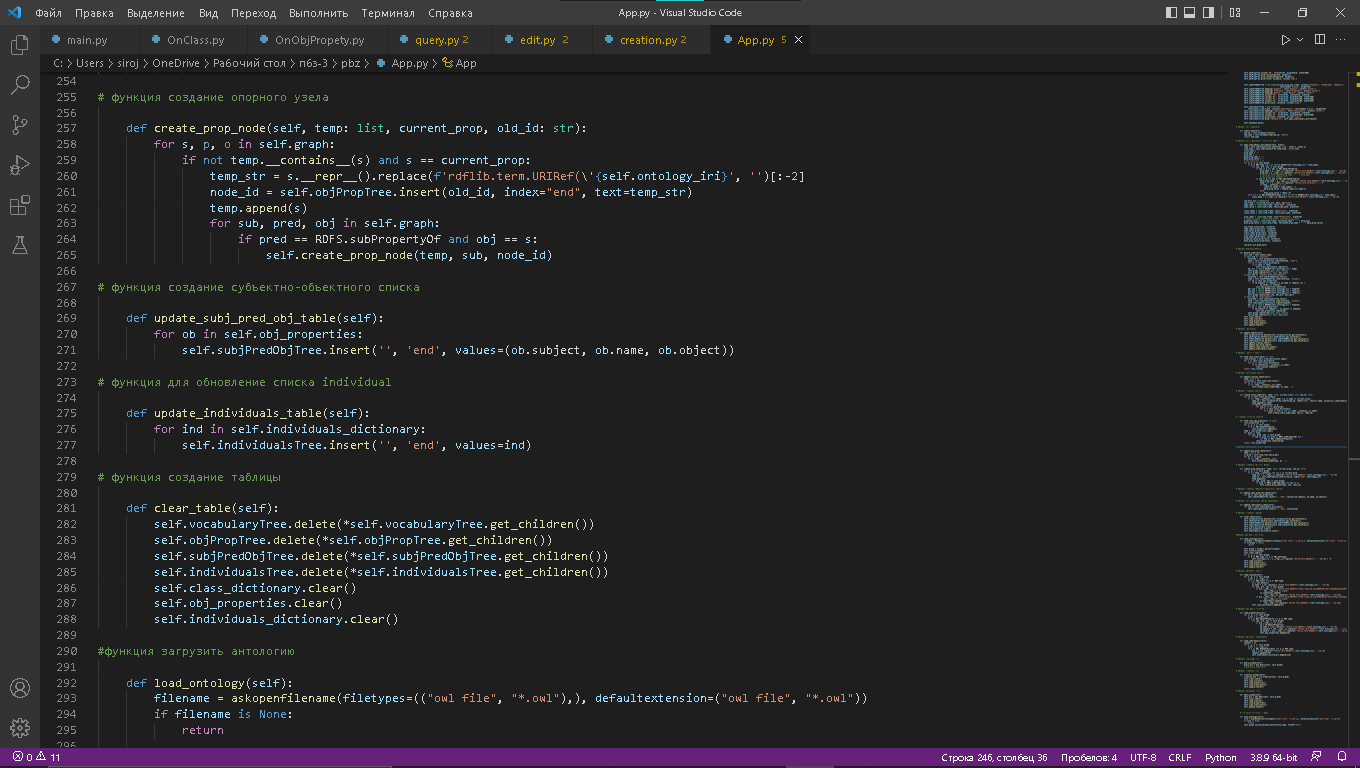


Рис .13

Создание опорного узела для классов функция создание кубъектно-объектного списка, функция для обновление списка individual и функция создание таблицы

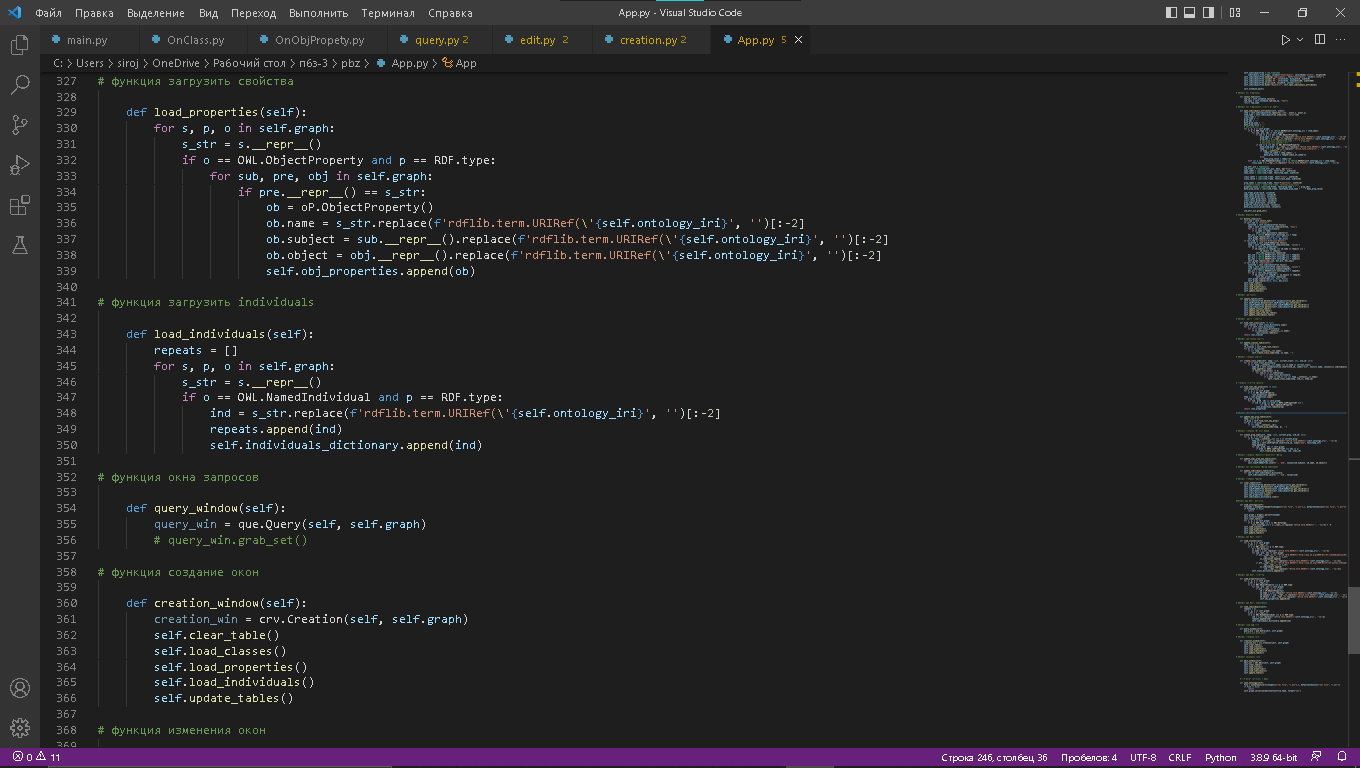


Рис. 14

Функции для работы с посторонней онтологией функция для загрузки свойств с другой онтологии функция для загрузки individuals функция для созранения онтологии , функции для создание запросов и функция для создание окон для запросов

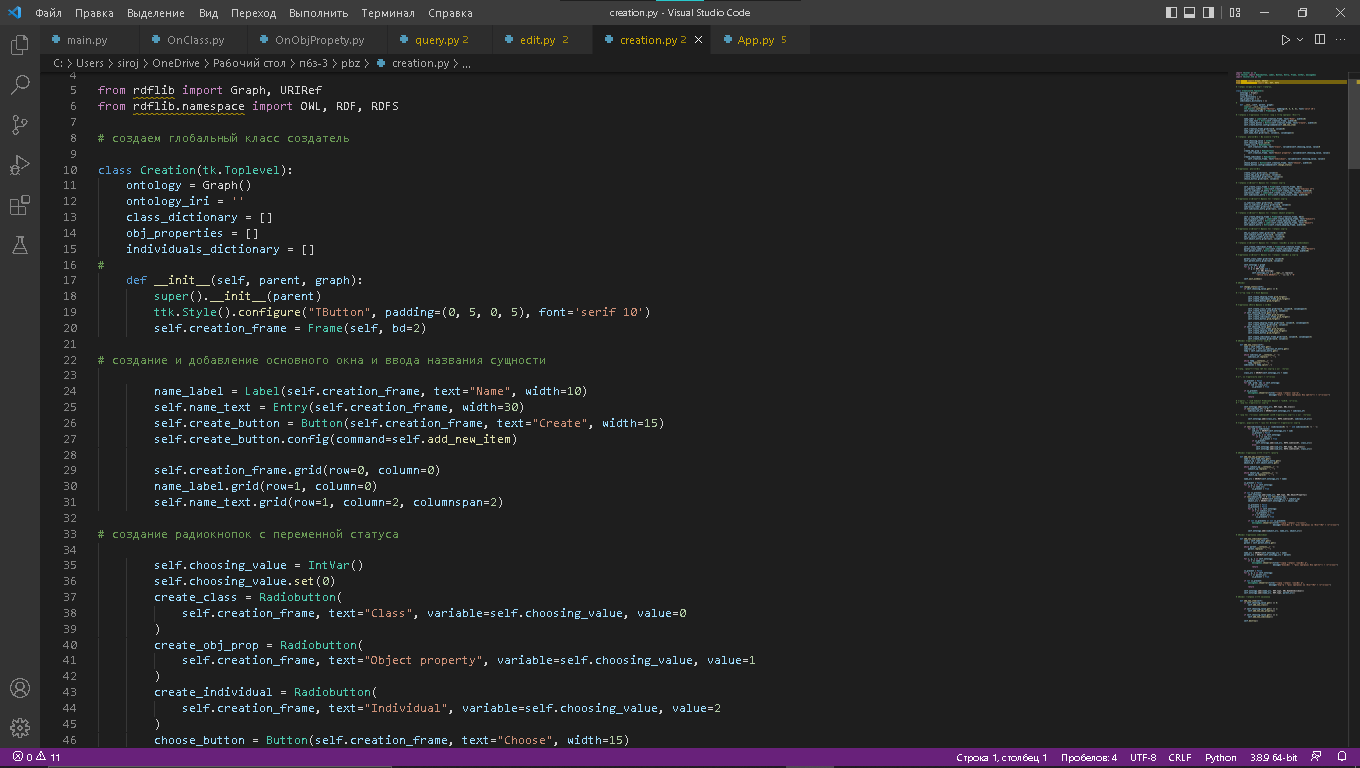


Рис. 15

Работа создание объектов и собъектов в онтологию создаем глобальный (Супер класс) Creation и создаем внутри нее окно для работы с объектами и радиокнопки с

переменной статуса

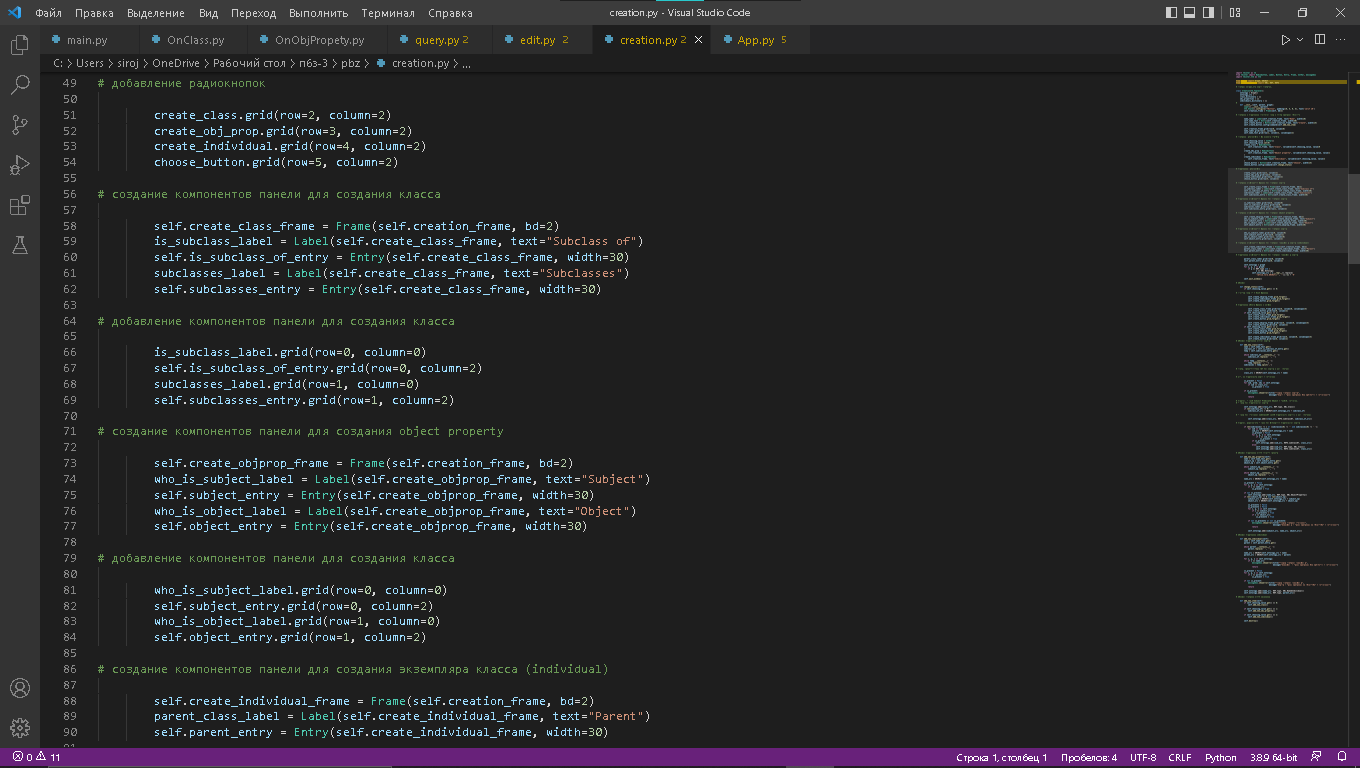


Рис. 16

Добавление радиокнопок для работы с классами создание компонентов для создание класса добавление компонентов для создание object property добавление компонентов панели для создание класса создание панели для содание компонентов для класса individual

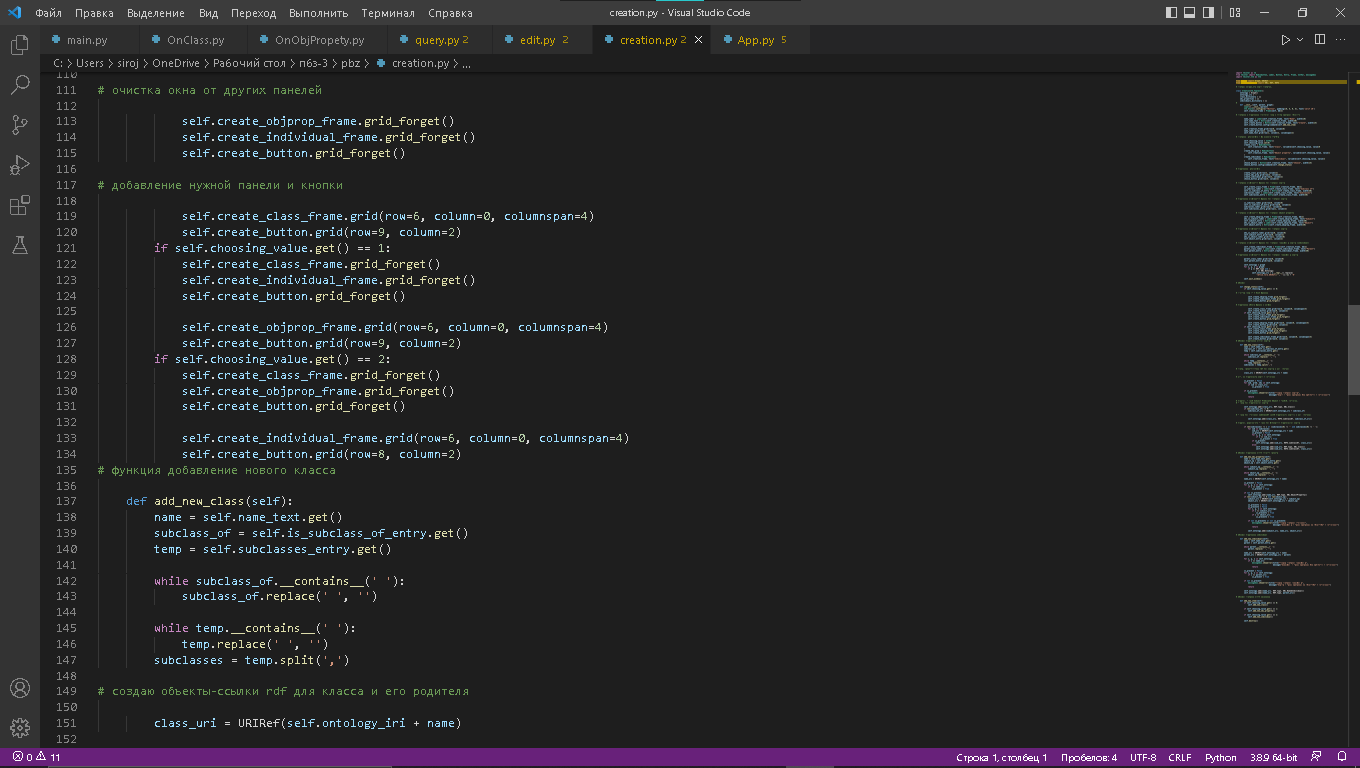


Рис. 17

Функции для добавления панели в онтологий и очистка панели онтологии вслед за добавлении панели идёт функция для добавление класса в панель

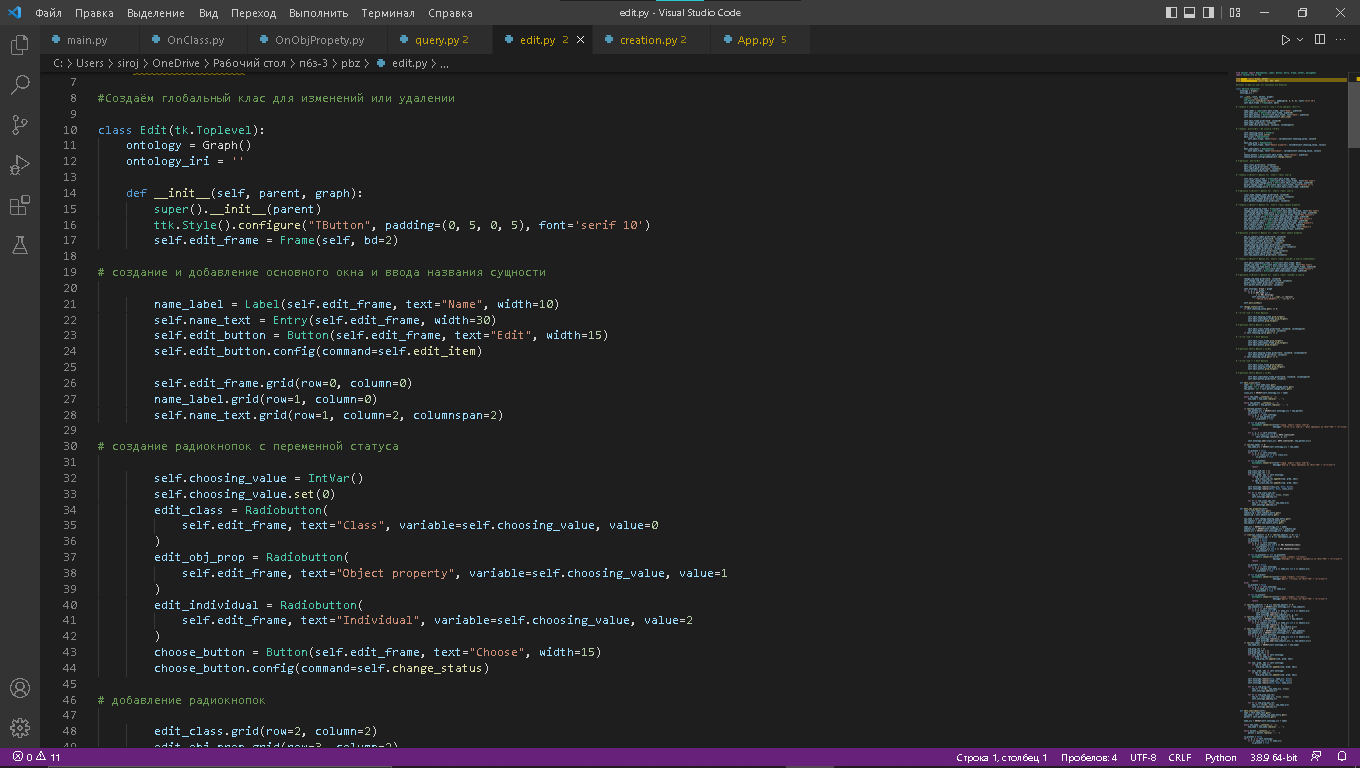


Рис. 18

Создаём супер клас для функции изменения и удаления антологии классов и individuals

Создание основного окна для ввода название сущности и создание радиокнопки с переменной статуса

И добавление радиокнопок

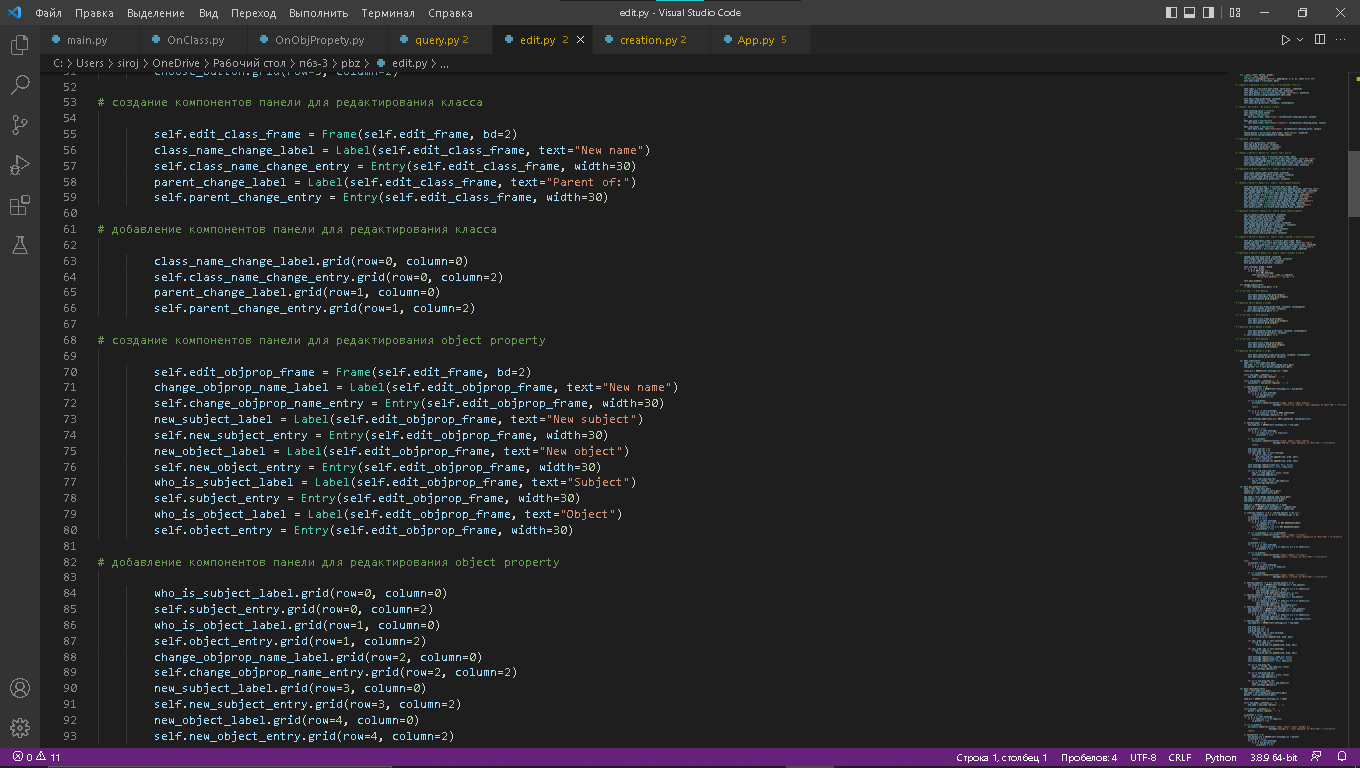


Рис. 19

Функция для создания компонентов панели для редоктирования класса и object property

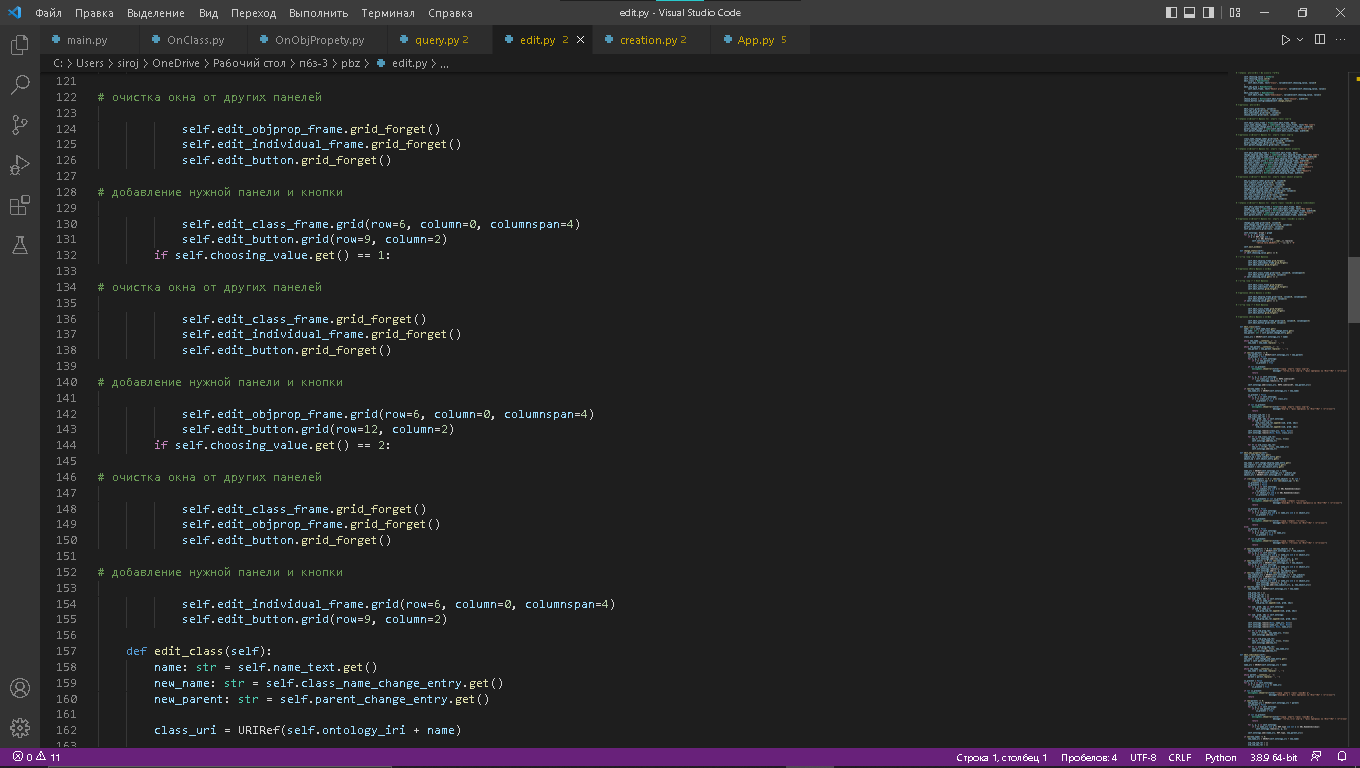


Рис. 20

Очистка окна от других панелей добавление кнопки лоя нужной понели

Очистка окна от других панелей , добавление нужной кнопки с панелем

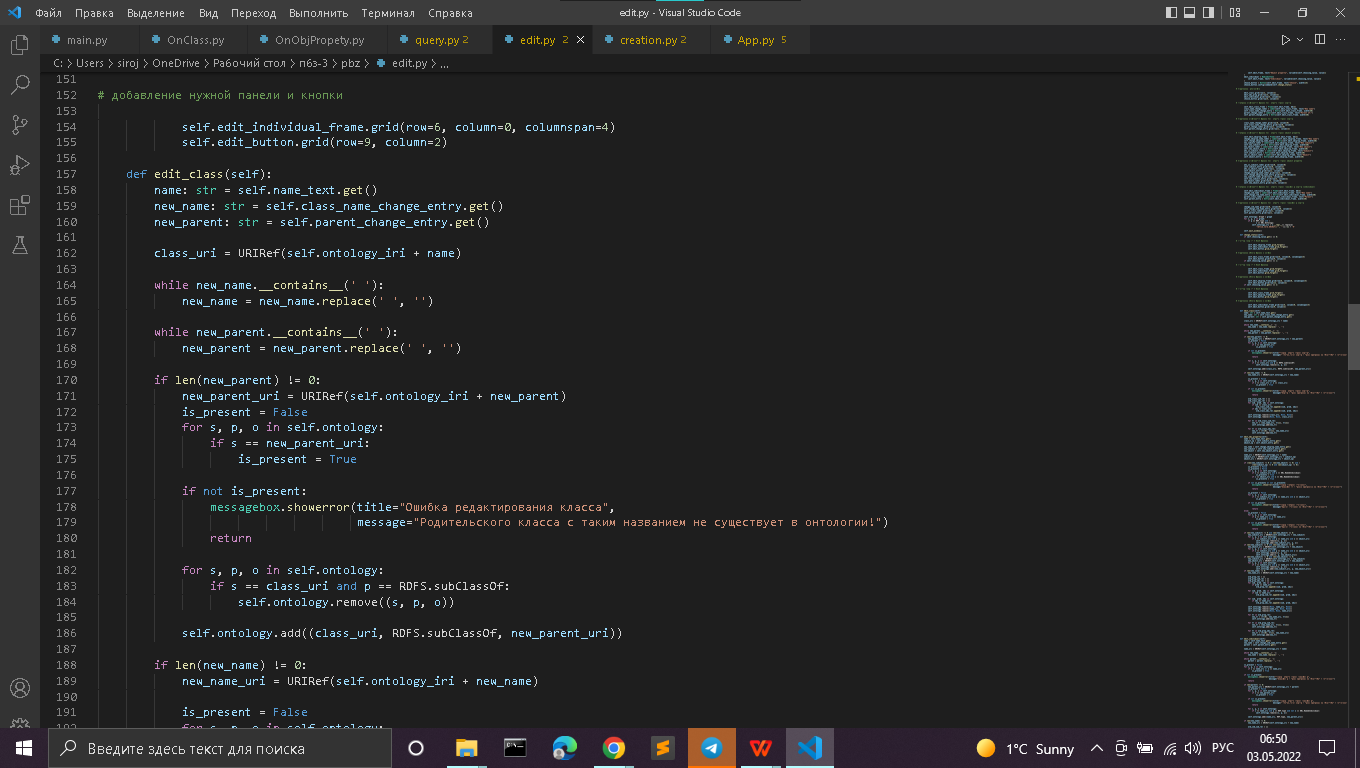


Рис . 21

Создаем класс для функций редоктирования

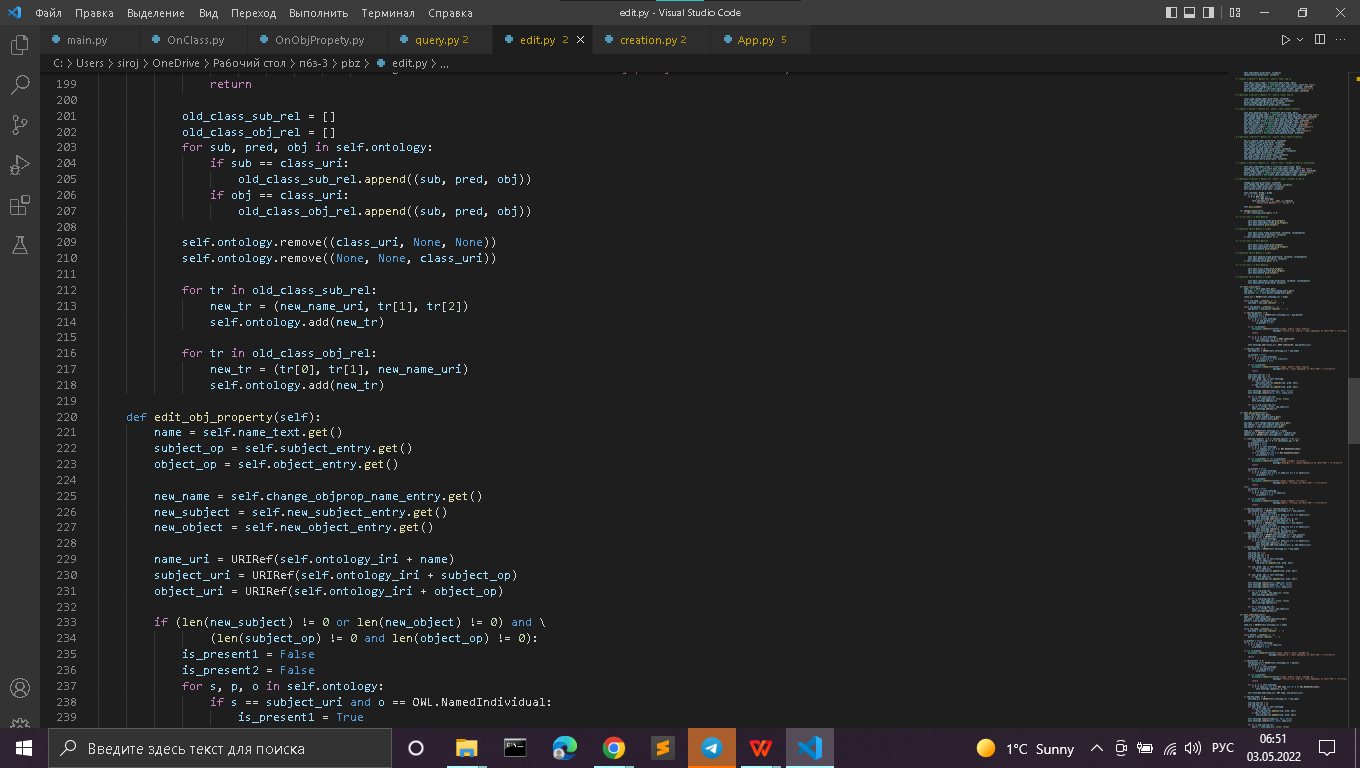


Рис. 21

Вносим в наш класс редоктирования все функция с rdfLib

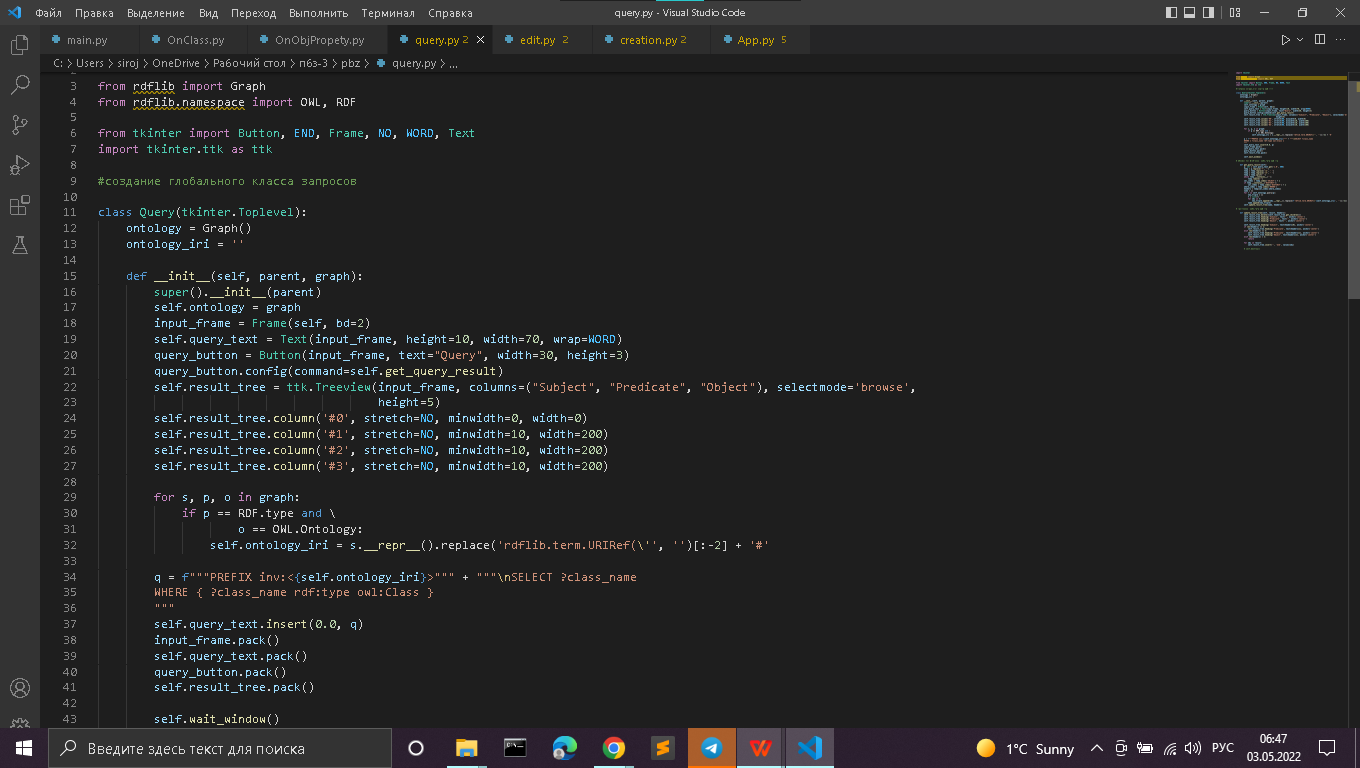


Рис . 22

Создаем навый файл query для запросов в этом файле в создадим функии и строения запросов query и SPRQL

Создаем супер клас query и вносим туда наши функии запросов

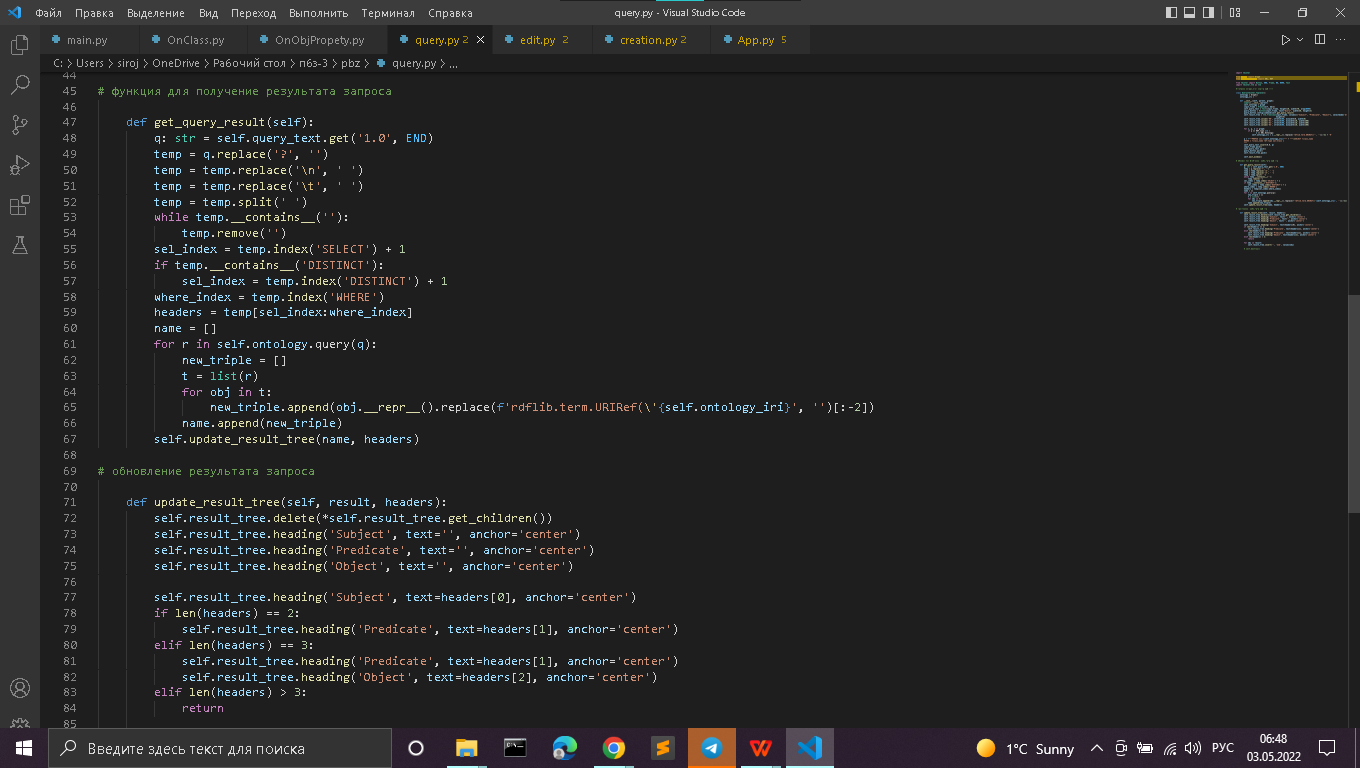


Рис. 23

Функция для получение результатов наших запросов и функция для обновление запросов

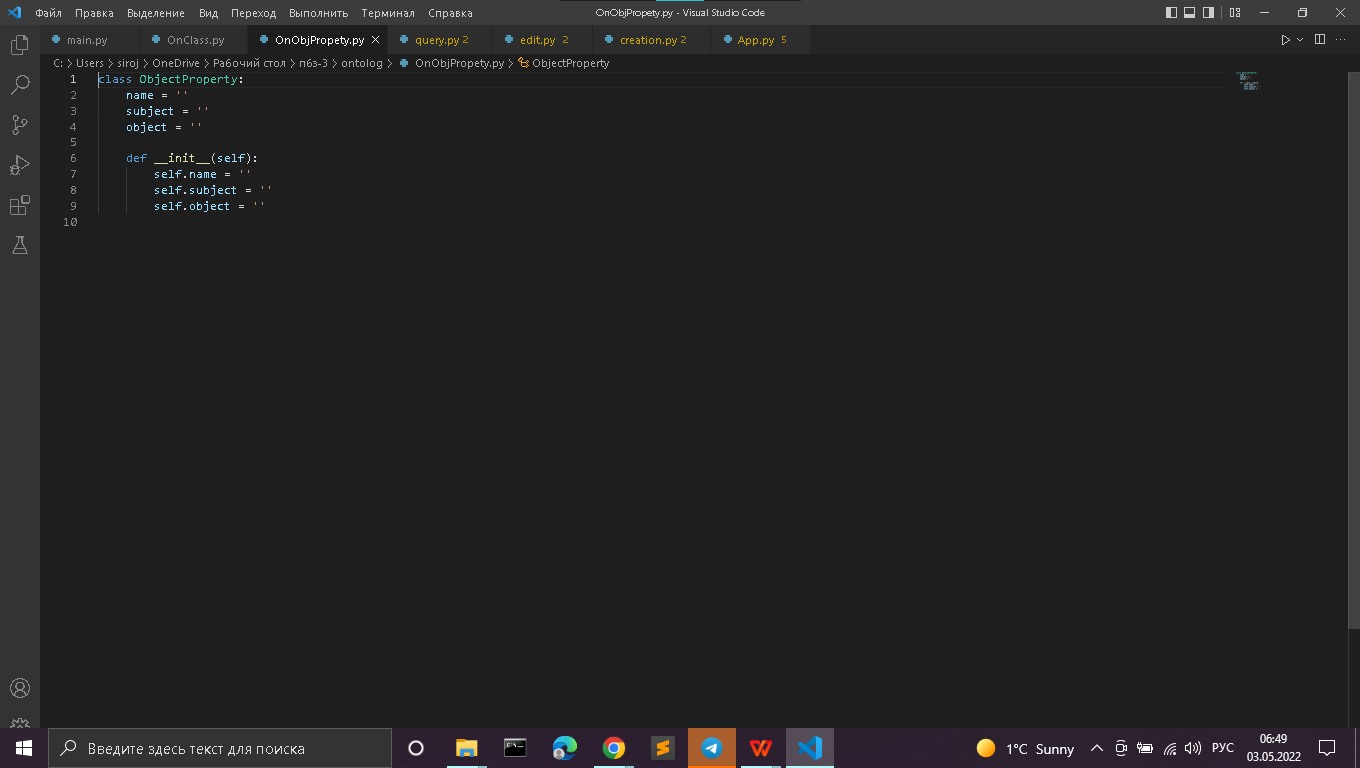


Рис 24

Создаем навый файл для свойства наших объктов и вносим свойства наших объектов

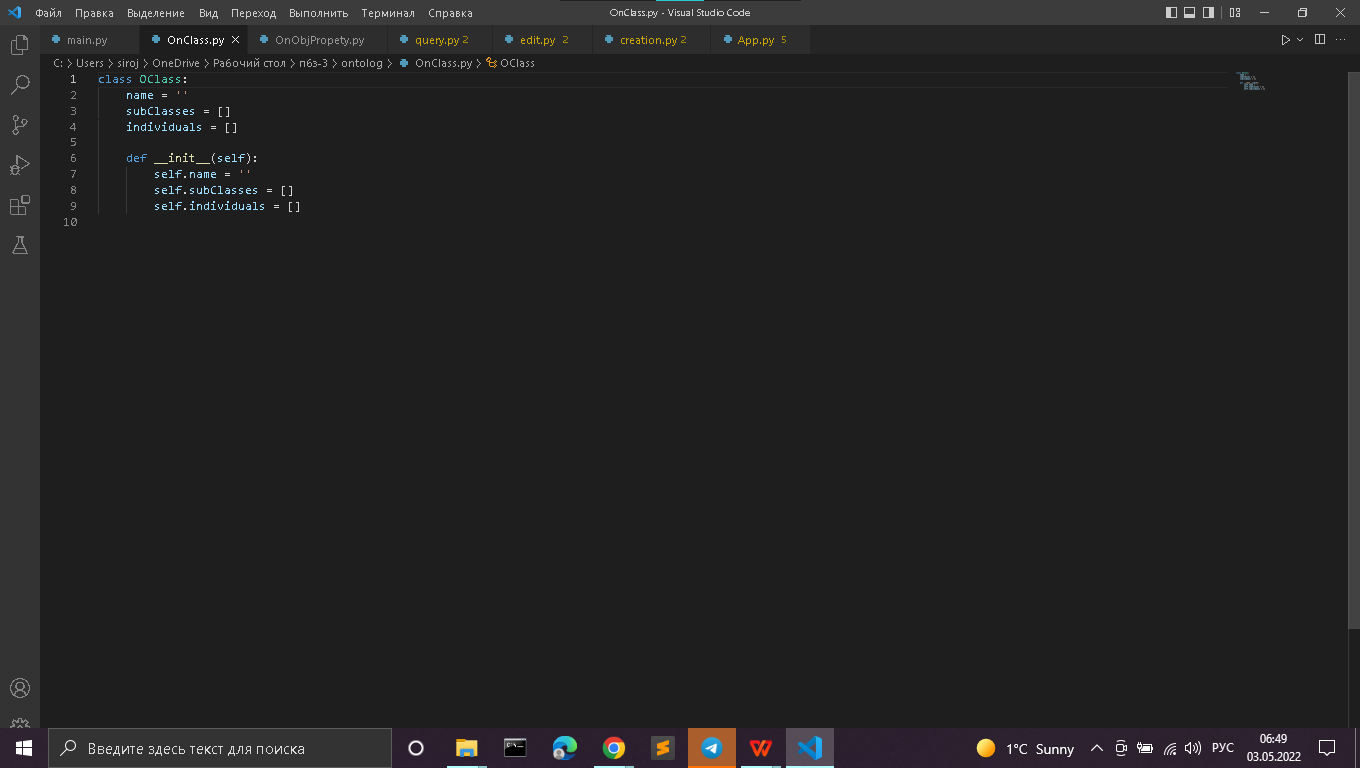


Рис 25

Создаём новый файл для создание параметров объекта и сам объект

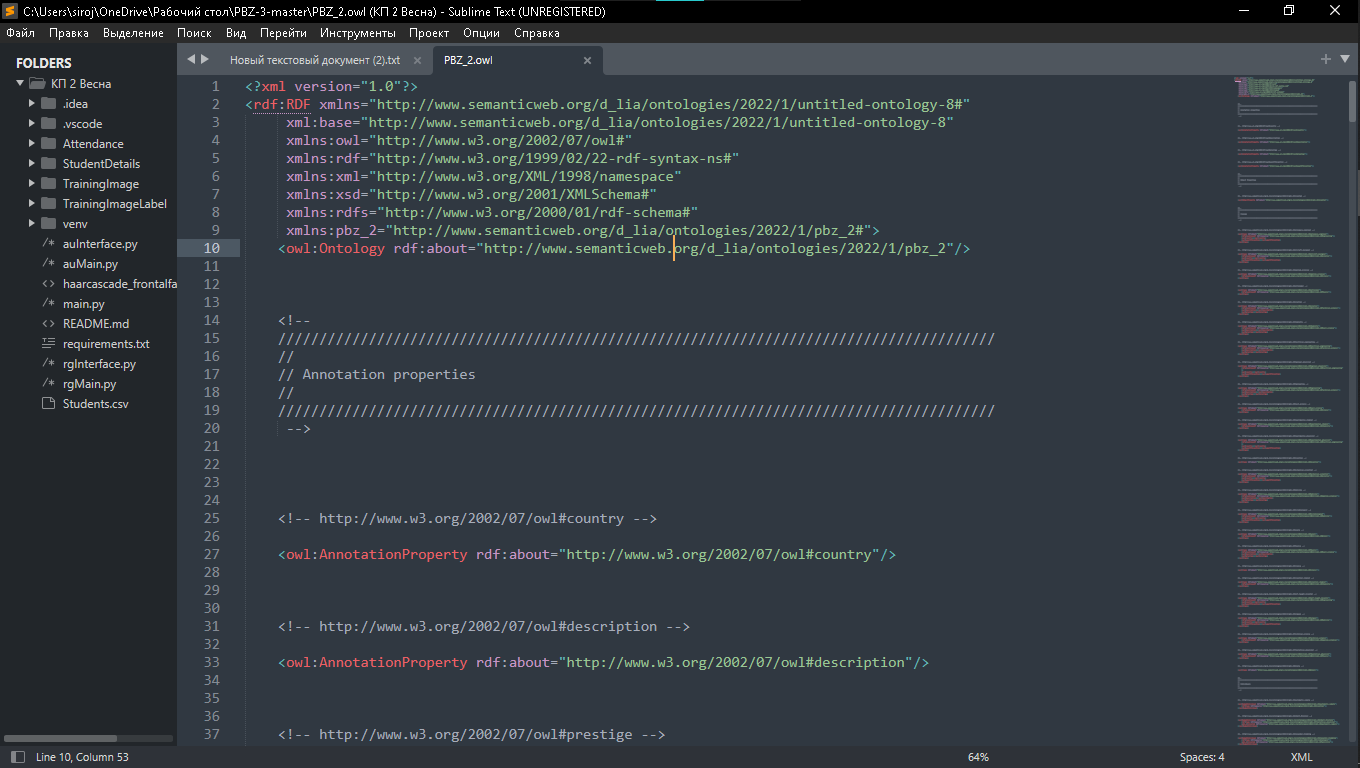


Рис. 25

Разработанный раздел “Search” посвящен поиску по нескольким направлениям - поиск по параметрам, поиск по родительскому классу, поиск по свободным объектам/классам. Поиск осуществляется с использованием языка запросов SPARQL, что в значительной мере упрощает работу.

Нами были реализованы функции сохранения и загрузки из файла. При сохранении вначале производится сохранение всех классов, потом всех объектов и только после этого - отношений. Для записи в файл берется понятие и выделяется соответствующими тегами.

Если рассматривать загрузку из файла, то схема ровно обратная - ищутся теги для классов, для объектов, для поиска отношений используется запросы языка SPARQL.

Ниже приведен пример RDF-файла, который сохраняем и загружаем в наше приложение. Все файлы имеют расширение .rdf.

**ОПИСАНИЕ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ**

В лабораторной работе мы погружали созданную нами онтологию локально. Для погружения мы использовали Jena API.

При выборе API мы опирались на ряд параметров, таких как совместимость с языком программирования, интегрируемость, удобство и простота установки. Ниже можно ознакомиться со всеми интересующими наш возможностями: техническая реализация, поддержка рекомендаций консорциума SPARQL World Wide Web (W3C) и доступные интерфейсы прикладного программирования (API).

При выборе API также учитывалась распространенность его документации, чтобы при работе с ним, в случае возникновения вопросов, было легко найти решение.

Apache Jena API- это платформа [семантической сети](https://en.wikipedia.org/wiki/Semantic_Web) с [открытым исходным](https://en.wikipedia.org/wiki/Open-source_software) кодом для [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) . Он предоставляет [API](https://en.wikipedia.org/wiki/API) для извлечения данных из графиков [RDF](https://en.wikipedia.org/wiki/Resource_Description_Framework) и записи в них. Графики представлены в виде абстрактной «модели». Модель может быть получена из данных из файлов, баз данных, URL-адресов или их комбинации. Модель также можно запросить

Virtuoso Universal Server - это гибрид [промежуточного программного обеспечения](https://en.wikipedia.org/wiki/Middleware) и [ядра базы данных](https://en.wikipedia.org/wiki/Database_engine), сочетающий в себе функции традиционной [системы управления реляционными базами данных](https://en.wikipedia.org/wiki/Relational_database_management_system) (СУБД), [объектно-реляционной базы данных](https://en.wikipedia.org/wiki/Object%E2%80%93relational_database) (ORDBMS), [виртуальной базы данных](https://en.wikipedia.org/wiki/Federated_database_system), [RDF](https://en.wikipedia.org/wiki/Resource_Description_Framework), [XML](https://en.wikipedia.org/wiki/XML), [свободного текста](https://en.wikipedia.org/wiki/Free-text) , [сервера веб-приложений](https://en.wikipedia.org/wiki/Application_server) и [файлового сервера.](https://en.wikipedia.org/wiki/File_server) функциональность в единой системе. Вместо того, чтобы иметь выделенные серверы для каждой из вышеупомянутых областей функциональности, Virtuoso является «универсальным сервером»; он позволяет использовать один [многопоточный](https://en.wikipedia.org/wiki/Thread_(computer_science)) серверный [процесс](https://en.wikipedia.org/wiki/Process_(computing)), который реализует несколько протоколов.

При более тщательном изучении мы пришли к выводу, что рациональнее всего использовать Jena API из-за большого количества дополнительной документации, примеров реализации и видео-уроков, также стоит заметить, что Jena API ориентирована на взаимодействие с кодом именно на Java (наличие уже реализованных библиотек).

**ВЫВОД**

В ходе выполнения данной лабораторной работы нами было создано приложение для работы с онтологиями, которое имеет следующий функционал: добавление, редактирование и удаление классов, объектов и отношений. Был реализован функционал позволяющий самостоятельно создавать SPARQL запросы: по одному, двум и трем параметрам, также есть поиск по объекту родительского класса и “свободному” объекту или классу.

Есть возможность как загружать уже созданную онтологию, так и сохранять новую. Для хранения онтологии было использовано RDF-хранилище,

**ИСТОЧНИКИ**

* https://python-scripts.com/tkinter
* https://ru.wikiversity.org/wiki/%D0%9A%D1%83%D1%80%D1%81\_%D0%BF%D0%BE\_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B5\_Tkinter\_%D1%8F%D0%B7%D1%8B%D0%BA%D0%B0\_Python#grid()[8]
* https://wiki.uib.no/info216/index.php/Python\_Examples
* https://wiki5.ru/wiki/Python\_(programming\_language)
* https://wiki5.ru/wiki/RDFLib