МАОУ «Классический лицей № 1» г. Ростов-на-Дону

**Создание поисковой системы для работы с иходными текстами на языке Питон с использованием представления кода в виде абстрактного синтаксического дерева**

Подготовил:

Батальщиков Вячеслав Александрович  
лицеист 10 ω класса

Руководитель:

Пусева Ольга Наиловна

2020 г.

# Введение

# Актуальность выбранной темы

Работа программистов связана с необходимостью чтения и поиска информации в больших объёмах программных текстов. С целью облегчения этих задач созданы множество инструментов, среди которых следует выделить два важных класса: интегрированные среды разработки (IDE) и системы контроля версий. И те, и другие предназначены для работы с конкретными проектами, однако, в практике программистов довольно часто возникают задачи поиска различных программных элементов (модулей, классов, функций) по широкой кодовой базе, включающей как собственные проекты, таки и системные и сторонние библиотеки и даже отдельные скрипты.

Отчасти указанная задача поиска информации решается интегрированными средствами общих хранилищ программных репозиториев, типа gitlab. Его можно использовать и для собственных разработок, развернув локально.

Однако иногда приходится иметь дело с кодом, не внесённым в систему контроля версий, причём вносить его может быть нецелесообразно (например, код стандартных библиотек) или трудозатратно (большое количество отдельных скриптов).

Представляемый проект призван решить отчасти эту проблему путём создания средства сканирования и поиска информации в кодовой базе на языке Python.

# Цель проекта Теоретические знания, практические навыки

Основная цель проекта -- создать полезное приложение для структурированного поиска информации.

В силу специфики выбранной задачи, удалось познакомиться с теоретическими основами работы компиляторов и интерпретаторов языков программирования, в частоности -- изучить работу начального этапа обработки программы, построение дерева синтаксического разбора.

Программная часть проекта представляет собой реализацию программы с графическим интерфейсом на языке Python, взаимодействующую с базой данных. Это позволило получить практические навыки в соответствующих областях: разработка пользовательского интерфеса с использованием библиотеки PyQT5 и работа с реляционными базами данных посредством библиотеки sqlite3.

Кроме этого, удалось повысить свой уровень владения языком программирования Python, в частности, получить опыт написания собственных функций-генераторов, работы с подсистемой журналирования и отладки сложных программ.

# Задачи проекта

Исходя из сформулированной цели была разработана архитектура проекта, включающая три основных компонента:

1. **База данных**, в которой храняться результаты обработки исходных тексотов.
2. **Сканер**, выполняющий непосредственную обработку входных файлов и помещающий рещультаты в базу данных.
3. **GUI** -- приложение с графическим интерфейсом, предназначенное для выполнения поисковых запросов к базе данных и наглядного представления результатов поиска.

Отталкиваясь от этой структуры, были сформулированы следующие задачи:

1. Выбрать подходящую базу данных и разобраться с основами работы с ней средствами языка Python.
2. Изучить декларативный язык запросов к реляционным системам баз данных SQL.
3. Изучить структуру данных абстрактного синтаксического дерева для языка программирования Python и научиться построению этого дерева и его обходу.
4. Познакомиться с моделью представления пользовательского интерфейса фреймворка QT и получить практические навыки разработки интерфейса пользователя посредством библиотеки PyQT5 для языка Python.
5. Предоставить программную реализацию программных модулей проекта.

## Разработка базы данных

В результате поиска подходящей библиотеки для работы с базами данных решено было остановиться на библиотеке sqlite3, предназначенной для работы с файлами баз данных формата SQLite. Это компактная встраиваемая СУБД со свободной лицензией и достаточно простым программным интерфейсом, благодаря чему она получила широкое распространение.

Далее нужно было определиться непосредственно со структурой базы данных. В рамках общей функциональности решено было ограничиться слеюущими элементами, извлекаемыми из файлов исходных текстов на языке Python:

1. глобальные переменные;
2. классы;
3. функции.

Каждый из этих элементов характеризуется именем и положением в исходном тексте -- номером строки.

Кроме самих извлечённых элементов, в базе данных очевидно следует хранить файловые пути к обработанным файлам, а также информацию о принадлежности извлечёнными элементам конкретным файлам. Исходя из этого, была разработана следующая структура таблиц:

1. таблица "files" для списка просканированных файлов
2. таблица "global\_vars(fname, name, line\_no)" для списка глобальных переменных
3. таблица "classes(fname, name, doc\_string, line\_no)" для списка классов
4. таблица "functions(fname, name, class\_name, doc\_string, line\_no)" для списка функций

В последних трёх таблицах поле "fname" -- внешний ключ к таблице "files".

База данных в формате SQLite представляет собой отдельных файл. Обычно в случае отсутствия файла базы данных принято создавать начальную базу данных «на лету» с помощью программного кода. Это было сделано и в данном проекте (scanner.py, функция create\_tables).

Идеологически, работы с базой данных организовывется путём группировки связанных изменений в единые элементы -- транзакции. Транзакция может либо быть полностью выполнена, либо полностью не выполнена, не допускется её частичное исполнение. Это сделано с целью обеспечения целостности базы данных. Типичная работа с базой данных в рамках библиотеки sqlite3 состоит из следующих шагов:

1. Подключение к файлу базы данных. Получаем объект connection.
2. Получение из connection объект cursor, который позволяет взаимодействовать с подключенной базой данных.

## Разработка сканера

Сканер предназначен для выполнения следующих основных задач:

1. Сканирование заданных файловых путей с целью поиска файлов с исходными текстами на языке Python (определяются по расширению «.py»)
2. Непосредственная обработка входного файла на языке Python, которая заключается в построении AST, обходе его и сборе описаний соответствующих программных элементов.
3. Сохранение найденных программных элементов в базу данных.

Обход файловой системы и поиск файлов с заданным расширением осуществлялся средствами стандартной библиотеки pathlib. Сохранение элементов в базу данных было рассмотрено в предыдущем пункте. Рассмотрим подробнее работу с абстрактным синтаксическим деревом.

## Абстрактное синтаксическое дерево

Согласно Википедии, в информатике абстрактное синтаксическое дерево определяется как граф специального вида: конечное помеченное ориентированное дерево, в котором внутренние вершины сопоставлены (помечены) с операторами языка программирования, а листья - с соответствующими операндами. Таким образом, листья являются пустыми операторами и представляют только переменные и константы.

Абстрактное синтаксическое дерево является связующим звеном между исходным текстом программы и структурой данных, которая затем используется в качестве внутреннего представления в компиляторе или интерпретаторе компьютерной программы для оптимизации и генерации кода.

Поскольку AST является графом, его можно наглядно представить - визуализировать. В качестве примера приведу разбора следующей программы (пример и визуализация взяты отсюда <https://ucb-sejits.github.io/ctree-docs/ipythontips.html> ):

import ctree

def f(a):

for x in range(10):

a[x] += x

tree1 = ctree.get\_ast(f)

ctree.ipython\_show\_ast(tree1)

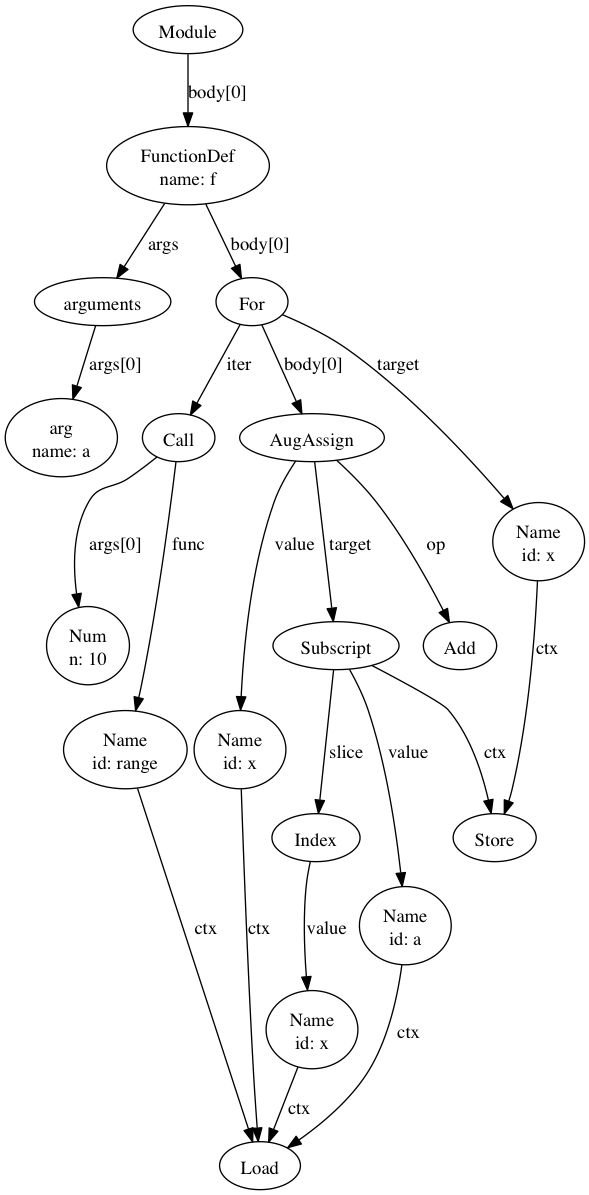


Рисунок 1. Визуализация дерева синтаксического разбора

Результат синтаксического разборка показан на Рисунке 1. Нас здесь интересует подчинённость сущностей, например аргументы функции (arguments) являются подузлом определения функции (FunctionDef). При этом нам доступны имена функций и аргументов, что является извлекаемой информацией в рамках нашей задачи.

Важной особенностью языка Python является наличие стандартной библиотеки для формирования AST, которая также называется ast. Благодаря этому удалось написать компактный код построения и обхода синтаксического дерева. По сути, извлечение информации из синтаксического дерева осуществляется одной сравнительно небольшой рекурсивной функцией, получаеющей на вход стартовый узел. Вот её код:

def walk\_tree(node, className=None, parent=None):

if isinstance(node, list):

for e in node:

yield from walk\_tree(e, parent=parent)

else:

if isinstance(node, ast.FunctionDef) or isinstance(node, ast.AsyncFunctionDef):

args = ','.join([a.arg for a in node.args.args])

docStr = ''

# print(type(node.body[0]))

if isinstance(node.body[0], ast.Expr) \

and isinstance(node.body[0].value, ast.Str):

docStr = node.body[0].value.s

# print(inspect.getmembers(node))

yield {

'type': 'func',

'class\_name': className,

'name': node.name,

'args': args,

'doc\_str': docStr,

'line\_no': node.lineno

}

if isinstance(node, ast.ClassDef):

className=node.name

docStr = ''

if isinstance(node.body[0], ast.Expr) \

and isinstance(node.body[0].value, ast.Str):

docStr = node.body[0].value.s

yield {

'type': 'class',

'name': node.name,

'doc\_str': docStr,

'line\_no': node.lineno

}

if parent is None and isinstance(node, ast.Assign):

for var in node.targets:

if not isinstance(var, ast.Name):

continue

yield {

'type': 'global\_var',

'name': var.id,

'line\_no': node.lineno

}

if hasattr(node, 'body'):

for e in node.body:

yield from walk\_tree(e, className=className, parent=node)

Из важных моментов, здесь сделует обратить внимание на проверку типа узла конструкцией вида isinstance(node, ast.Assign), а также на выражение с ключевым словом yield, которое означает, что данная функция является генератором (в терминологии языка Python), то есть она возвращает промежуточные значения и может быть использована во внешнем коде подобно обходу коллекции, где в качестве коллекции выстпает результат вызова функции-генератора.

## Создание GUI приложения для поиска информации

Для диалога с пользователем я использовал фреймворк PyQt5. Этот модуль работает с ui файлами -- файлами деклоративной разметки, в которых содержится информация о расположении объектов и их параметров в приложении.

Класс основного окна MainWindow, унаследованный от QMainWindow, представляет главное окно, в котором пользователь вводит параметры поиска в окне в объект QLineEdit, а тип сущности, которую ищет пользователь, задаётся с помощью QRadioButton. Затем пользователь нажимает кнопку и открывается окно с таблицей, в которой содержится строчка и путь к файлу, сущность в котором соответствует параметрам поиска. Если нажать на путь файла, откроется окно с программой на строчке с искаемой сущностью.

## Заключение

Выполнение проекта позоволило достичь заявленные цели. Прежде всего это касается получения теоретических знаний и практических навыков. Удалось продвинуться в понимании устройства компиляторов и баз данных, познакомиться с моделью асинхронных вычислений, ярким примером которой является типичное GUI приложение.

В реализации присутствуют определённые недостатки. Так, по результатам тестового использованпия, модуль сканирования хочется расширить поддержкой имён аргументов функций и аннотаци1 типов (там где они есть). Модуль GUI также несомненно требует доработки в плане эргономики и расширения возможностей поиска (например, поиск по регулярным выражениям). Но в целом, разработанное приложение показало работоспособность исходной идеи: сканер показывает удовлетворительную скорость обработки файловой системы, а база данных SQLite выдерживает практически значимые объёмы.

Хочется надеяться, что проделанная работа получит дальнейшее развитие.

## Источники информации