**Knowledge Graph Generation**

**Introduction**

This project aimed to create a knowledge graph from source code files and import it into a Neo4j database. The project consists of two main components: code analysis to generate a knowledge graph and the import of this graph into a Neo4j database. This report provides an overview of the project's objectives, methodologies, and outcomes.

**Objectives**

The primary objectives of this project were as follows:

1. Analyze code files written in Python and MATLAB to extract relationships between entities.

2. Create a knowledge graph representing these relationships.

3. Import the knowledge graph into a Neo4j database for querying and visualization.

**Code Analysis**

Knowledge Graph Generation

The code analysis component involves parsing source code files written in Python and MATLAB to extract relationships between entities. The relationships include function calls, object creations, variable assignments, class definitions, module imports, dataflow, database connections, and API calls. The extracted relationships are categorized based on their type.

For Python code:

- Function calls are identified and labeled as "function."

- Object creations are labeled as "function."

- Variable assignments are labeled as "variable."

- Class definitions are labeled as "class."

- Module imports are labeled as "module."

- Dataflow relationships are labeled as "function."

- Database connections are labeled as "database."

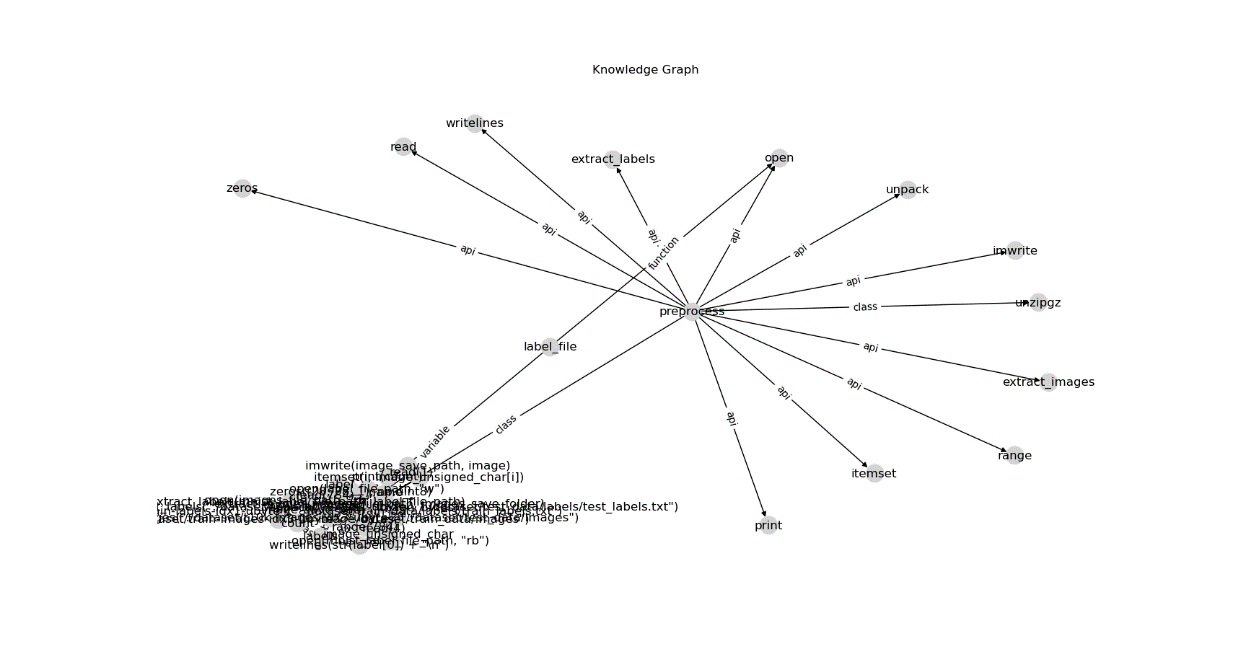
- API calls are labeled as "api."

For MATLAB code:

- Class definitions are labeled as "class."

- Function modules are labeled as "function."

The relationships are then used to construct a directed graph using NetworkX.



**Neo4j Database Import**

The Neo4j database import component involves taking the generated knowledge graph and importing it into a Neo4j database for further analysis.

- A Neo4j driver is used to connect to the Neo4j database.

- The knowledge graph data is read from a text file (knowledge\_graph.txt).

- The data is split into lines, and each line is processed to create nodes and relationships in the Neo4j database.

A customized import logic is used to match the knowledge graph's data structure. The Cypher query language is utilized to create nodes and relationships in the Neo4j database. The query includes "MERGE" statements to ensure that duplicate nodes are not created.

**Conclusion**

This project successfully achieved its objectives by extracting relationships from source code files, constructing a knowledge graph, and importing it into a Neo4j database. The resulting knowledge graph can be used for various applications, including code analysis, visualization, and querying.

The project's modular design allows for flexibility in handling different code languages and knowledge graph structures. Future enhancements may include additional code language support, more complex relationship extraction, and improved data import processes.

This project demonstrates the potential of knowledge graphs in organizing and visualizing complex relationships within codebases, making it a valuable tool for software developers and analysts.

For further information or to access the code used in this project, please refer to the project's GitHub repository.