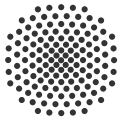

University of Stuttgart

Programming Paradigms Prof. Dr. Michael Pradel

University of Stuttgart, Summer 2023

Exercise 1: Scanners and Parsers

(Deadline for uploading solutions: May 4, 2023, 11:59pm Stuttgart time)

The materials provided for this homework are:

- a pdf file with the text of the homework (this document);
- a zip file with **the templates file and folder structure you must use for the submission**.

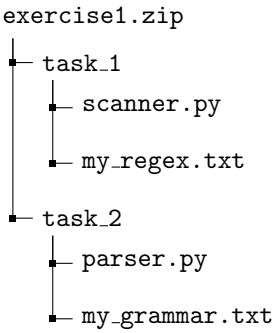

Figure 1: Folder structure to use for submission.

The submission must be compressed in a zip file (No ~~rar~~, ~~7z~~, ~~gz~~...) using exactly the folder structure and names in Figure 1.
The assignment should be implemented in **Python (version >= 3.8)**

Note: Submissions that do not comply with the exact folder structure and names in Figure 1, or that do not work with Python (version >= 3.8), will not be graded.

1 Task 1 (40% of total points of the exercise)

In this task, you have to create a scanner that converts a string into a sequence of tokens, or returns `None` value when impossible. You must manually implement the scanner; do not utilize a scanner generation tool. Third-party Python packages are also not permitted. Only the basic Python packages that come with the Python installation are permitted.

The tokens of the target language are expressed as regular expressions. Each student gets their own set of regular expressions, i.e., every student has a task that is slightly different from the others. You can download your regular expressions from `https://grammar-generator.anvil.app` under Task 1 page. The downloaded file is called **my_regex.txt**. Include the file in your ZIP file while adhering to the format shown in Figure 1. Furthermore, do not alter the file's content. The file begins with an identification number in the header, then the definition of the set of regular expressions, and lastly a set of input-output examples.

Spaces in the input string are allowed between tokens and should be skipped during scanning. That is, these characters do not cause an error and they should not appear in the output token sequence. Note that a scanner does not check whether an input string parses into a valid parse tree or abstract syntax tree. All it does is to split the input string into a sequence of valid tokens (or determine that this is impossible).

1.1 Instructions

1. Download your set of regular expressions from `https://grammar-generator.anvil.app` and move it into the folder task_1 contained inside the exercise zip file.
2. Use the provided template (`scanner.py`) to implement the scanner.
3. Test your scanner using the examples given in the downloaded file (`my_regex.txt`) by calling the script `scanner.py` and passing `my_regex.txt` as an argument (e.g., by running a command similar to this: `python3.9 scanner.py my_regex.txt`). The scanner will print the results into your terminal and save them into a json file in the same directory where you call the script from. In addition, you can save the printed output into a text file by executing a command similar to: `python3.9 scanner.py my_regex.txt > output.txt`

1.2 Structure of my_regex.txt

The downloaded file contains three sections separated by a series of '=' characters. The first section is the header containing an identifier which you can ignore. Second, you find the list of regular expressions for which you should implement a scanner. The third section is a set of possible input-output pairs. Input-output pairs are separated by a series of '-' characters. Each input-output pair takes two lines, one line for the input and the second one for the output.

1.3 Evaluation

Your code will be tested against a secret set of test inputs. We will run your Python script in a fresh and new Python environment that contains only standard Python packages. We do not permit the installation of new packages, therefore make sure you do not use any dependencies that are not included in the standard Python library.

2 Task 2 (60% of total points of the exercise)

In this task, you have to create a parser that analyzes a given string and decides whether it is accepted by a given grammar. You must manually implement the parser; do not utilize a parser generation tool. Third-party Python packages are also not permitted. Only the basic Python packages that come with the Python installation are permitted.

Similar to Task 1, each student gets their own grammar. The grammar is *not* related to the tokens used in Task 1. Please download your grammar from `https://grammar-generator.anvil.app` under Task 2 section. The downloaded file is called **my_grammar.txt**. Include the file in your ZIP file while adhering to the format shown in Figure 1. Furthermore, do not alter the file's content. The file begins with an identification number in the header, then the definition of the rules of the grammar, and lastly a set of input-output examples. **The given input is already tokenized and the tokens are separated by a space character.**

2.1 Instructions

1. Download your grammar from `https://grammar-generator.anvil.app` and move the downloaded file into the folder task_2 contained inside the exercise zip file.
2. Use the provided template (`parser.py`) to implement the parser.
3. Test your parser using the examples given in the downloaded file (`my_grammar.txt`) by calling the script `parser.py` and pass `my_grammar.txt` as an argument (e.g, by running a command similar to this: `python3.9 parser.py my_grammar.txt`). The parser will print the results into your terminal and save them into a json file in the same directory where you call the script from. In addition, you can save the printed output into a text file by executing a command similar to: `python3.9 parser.py my_grammar.txt > output.txt`

2.2 Structure of my_grammar.txt

The downloaded file contains four sections separated by a series of '=' characters. The first section is the header containing an identifier which you can ignore. Second, you find the grammar rules for which you should implement a parser. The third section is a set of possible input-output pairs. Input-output pairs are separated by a series of '-' characters. Each input-output pair takes two lines, one line for the input and the second one for the output.

2.3 Evaluation

Your code will be tested against a secret set of test inputs. We will run your Python script in a fresh and new Python environment that contains only standard Python packages. We do not permit the installation of new packages, therefore make sure you do not use any dependencies that are not included in the standard Python library.