UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

WEB PYTHON INTERPRETER WITH SPECIFYING THE REQUIREMENTS

Diplomová práca

2018

Bc. Ákos Hervay

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

WEB PYTHON INTERPRETER WITH SPECIFYING THE REQUIREMENTS

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	prof. RNDr. Andrej Blaho, PhD.

Bratislava, 2018 Bc. Ákos Hervay

Poďakovanie

Touto cestou by som sa chcel v prvom rade poďakovať ...

Abstrakt

Todo

Kľúčové slová: compiler, python, javascript

Abstract

This paper describes the analysis, proposal and the implementation of an web based programming environment written in Javascript for editing and running Python programs. The language implemented is a subset of the Python3 programming language, what could be used in the teaching of programming in a primary or high school environment. It would allow the teacher to give the students assignments with a predetermined set of limitations for the language, like the number of variables, allowed data types or the usage of standard methods.

Keywords: compiler, python, javascript

Contents

1 Introduction **9**

2 Analysis **11**

2.1 Existing applications 11

2.1.1 Trinket 11

2.1.2 Skulpt 12

2.1.3 Repl.it 12

2.2 Conclusion 14

2.3 Compilers 14

2.4 Parsers 16

2.5 Parser generators 17

3 Proposal **19**

3.1 Language specification 20

3.2 Compiler 21

3.2.1 Rules 21

3.3 Interface 23

3.3.1 Student view 23

3.3.2 Teacher view 23

4 Implementation **24**

4.1 Backend 24

4.1.1 Compiler 24

4.1.2 Database 24

4.2 User Interface 24

4.2.1 User - Student 24

4.2.2 User - Teacher 25

5 Conclusion **26**

5.1 Comparison 26

Chapter 1

Introduction

As the schooling of Python is spreading into primary and high schools, there could be a place for a more specialised environment for teaching the methodologies of programming. In these settings the students are exposed only a subset of the features that a language can provide.

It could be beneficial to have a tool for the teacher, with what he can use to create dedicated assignments where the students are limited by a set of parameters. These can help them to develop the programming principle or feature of the language that the task is trying teach to and maybe guide them in solving the task much more easily.

This is the problem thesis is wishes so solve, is the implementation of a web based interpreter for the programming language Python. The main requirement is that the application could be configured in a secondary interface so that specific features of the language can be disabled for the selected instances.

For this purpose I have analysed multiple already existing applications with similar set of features. Then I will outline a proposal of a web based Python interpreter. This is followed by the description of my implementation of this application. I will conclude this with a comparison of the application and the investigated programs, and a short evaluation of it's viability.

Chapter 2

Analysis

While in my research for this paper I have not found an application that covers the same exact use-case as described in my proposal, I have found multiple ones with the same or similar subset of features. There were multiple in-browser implementations of compilers. I have searched mainly for Python, but some also supported multiple languages.

2.1 Existing applications

2.1.1 Trinket

This application is the most similar one to my implementation. As it's main feature, with it you can create a *trinket*[1], which is an interactive code snippet that can be executed in the browser. It also provides platform for creating courses. The courses have a simple structure, with a list of topics, where each topic can have multiple pages. These pages are written in a simple *Markup* style editor. It supports the embedding of links, images, videos and runnable code snippets.

[1]https://trinket.io/

They have implemented multiple languages for creating trinkets like Python, R, HTML, Java. As this service is mainly for education it provides a Scratch[2]-like block based editor. These ease the students new to programming into the thought process of programming. It has also a canvas, on which in Python one can draw with the help of popular libraries like turtle, pygal or mathplotlib. With the drawing on a canvas they can familiarise themselves with the capabilities of the language.

As I plan to support a similar set of features to the one of this application. While the creation of documents is well executed, the design of it is intuitive and practical, in my opinion is a bit limited. While I have seen a couple of exercise that have build in test functionality, I have not found a way to create them in the documents.

2.1.2 Skulpt

Skulpt[3] as it is stated on its web page, is an open source in-browser implementation of Python 2.x. It has an interactive REPL and has support for code debugging. It also provides a way to make external libraries available to use for development from an external source. This library is used by the Trinket application to run the Python code.

2.1.3 Repl.it

Repl.it[4] is a much more mature version of Trinket. It is an open-source, cloud based interactive programming environment, with a support for a plethora of languages, form the regular ones like C, Java, Ruby and Python

[2]https://scratch.mit.edu/
[3]http://www.skulpt.org/
[4]https://repl.it/

to web-based like JavaScript, HTML or CSS. It even has support for building React native application for mobile environment. It has a fully featured code editor, with support for debugging for some languages. The compilation of the code is done on the server side on a virtual machine. Given the programming language has support for it, the the application provides an interactive console for code execution. Being that the written code is executed on a virtual machine, existing libraries can be imported the same way one would do in a desktop environment.

The feature that I focus on is the classroom service they provide. It is a product where one can register as a teacher, and can create a Classroom group for students to enroll to, where he can publish assignments in the supported languages. These assignments consist of the description what can be edited with a WYSIWYG[5] style editor, a code editor where the teacher can provide some starting sample code, and an interactive console. Optionally teacher can set a due date for the assignment, what is then can be tested with automatic Unit tests (language dependent), with matching of the input/output strings regular expressions, or manually. For assignments using the automatic unit tests, there can be added an model solution for those students who completed the assignment. An assignment can be set to be dependent of the previous ones, constraining the students to complete those before continuing.

This service further provides a way for teachers to collaborate on creating assignments, provided they are subscribed to their premium plan. Additionally, these classroom environments can be shared online trough their Community service or cloned from GitHub.

This part of the application is what I will model my implementation on, extending on the assignment structure, with a way to score each students

[5]What you see is what you get

progress. While my program will not be cloud based, the testing of the assignments will be done server side, in consideration of the possibility for the solution to be accessed otherwise.

2.2 Conclusion

In light of my review findings I conclude that while there are applications that have the features similar proposed in this work, with the addition of additional functions it is possible to enhance the user experience and usability of such programs.

2.3 Compilers

While my analysis of the goal of this document, I have found that while there are multiple libraries that could be used to compile Python on the web (like the one mentioned in the previous section), the implementation of a compiler would be necessary. In light of that I have set on to research what type of compilers there exist and what kind should I implement.

A compiler is computer software that transforms computer code written in one programming language (the source language) into another programming language (the target language). The name compiler is primarily used for programs that translate source code from a high-level programming language to a lower level language (e.g., assembly language, object code, or machine code) to create an executable program.[6]

In the design of a compiler there are multiple stages. These include a front end, a middle end, and a back end. The front end analyses the source code to build an internal representation of the program, and also manages the symbol table, a data structure mapping each symbol in the source code to associated information such as location, type and scope. This is commonly split into several phases which may execute sequentially or concurrently. Most commonly today, the front-end is broken into three phases: lexical analysis, syntax analysis, and semantic analysis. In some cases additional phases are used, notably line reconstruction and preprocessing, but these are rare.

The middle end performs optimizations on the intermediate representation in order to improve the performance and the quality of the produced machine code.[1] The middle end contains those optimizations that are independent of the CPU architecture being targeted. The back end is responsible for the CPU architecture specific optimizations and for code generation.

Lexical analysis

Lexical analysis (also known as lexing or tokenization) is the process of breaking the source code text into a sequence of small pieces called lexical tokens. This can be further divided into the *scanning* procedure which segments the input text into syntactic units called lexemes and assign them a category; and the evaluating, which converts lexemes into a processed value. A token is a pair consisting of a token name and an optional token value.[1]

Common token categories may include identifiers, keywords, separators, operators, literals and comments, although the set of token categories varies in different programming languages. The lexeme syntax is typically a regular language, so a finite state automaton constructed from a regular expression can be used to recognise it.

Syntactic analysis

Syntax analysis (also known as parsing) involves parsing the token sequence to identify the syntactic structure of the program. This phase typically builds a parse tree, which replaces the linear sequence of tokens with a tree structure built according to the rules of a formal grammar which define the language's syntax. The parse tree is often analysed, augmented, and transformed by later phases in the compiler.[1]

Semantic analysis

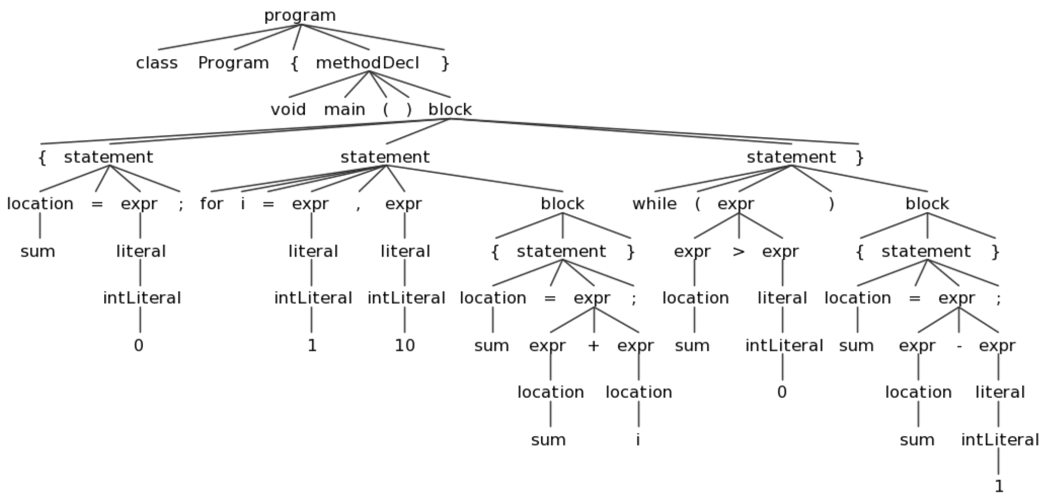
Semantic analysis adds semantic information to the parse tree and builds the symbol table. This phase performs semantic checks such as type checking (checking for type errors), or object binding (associating variable and function references with their definitions).

2.4 Parsers

The task of the parser is essentially to determine if and how the input can be derived from the start symbol of the grammar. This can be done in essentially two ways:

Top-down parsing - Top-down parsing can be viewed as an attempt to find left-most derivations of an input-stream by searching for parse trees using a top-down expansion of the given formal grammar rules. Tokens are consumed from left to right. Inclusive choice is used to accommodate ambiguity by expanding all alternative right-hand-sides of grammar rules.[1]

Bottom-up parsing - A parser can start with the input and attempt to rewrite it to the start symbol. Intuitively, the parser attempts to locate the most basic elements, then the elements containing these, and so on. LR parsers are examples of bottom-up parsers. Another term used for this type of parser is Shift-Reduce parsing.



program
class Program { methodDecl }
void main ( ) block
{ statement statement statement }
location = expr ; for i = expr , expr block while (expr) block
sum literal literal literal { statement } expr > expr { statement }
intLiteral intLiteral intLiteral location = expr ; location literal location = expr ;
0 1 10 sum expr + expr sum intLiteral sum expr - expr
location location 0 location literal
sum i sum intLiteral
1

Figure 2.1: Example of an abstract-syntax tree representation of a for loop

One of the examples for a top-down parser is the LL parser which appliable for a subset of context-free languages. It parses the input from Left to right, performing Leftmost derivation of the sentence. An LL parser is called an LL(k) parser if it uses k tokens of lookahead when parsing a sentence. LL grammars, particularly LL(1) grammars, are of great practical interest, as parsers for these grammars are easy to construct, and many computer languages are designed to be LL(1) for this reason.[7] And between those languages that is LL(1) is also Python.

2.5 Parser generators

In computer science **parser generators** (or compier-compiers) are programming tools that create a parser, interpreter or compiler from a predefined formal description of a system or language. This is mostly defined in a textual format with the grammar rules of the syntax of a programming language written in **Backus–Naur form**(BNF). The generated output mostly is some source code of the parser for the programming language. The ones that I have researched are ANTLR[6] and PLCC[3].

However I have decided against the use of such tools. While they are practical, the use of them notably accelerates the speed of development, in my opinion they do not provide the flexibility that the requirements of this thesis need. Besides I do not have the experience needed in the use of them. Further I would like to gain more experience in the developement of compilers and with the use of instruments like this, I would not gain less.

[6]http://www.antlr.org/

Chapter 3

Proposal

In this document I propose the implementation of a web based application for creating assignments for students, with what they can solve them in the browser without requiring to download anything.

The application has both server side and client side components. Client side is the main part of the application while the server only handles the storage. The client side can be divided into two parts: user/student and manager/teacher.

The teacher's dashboard contains all the necessary tools the create new and configure already existing exercises. An exercise is a nested set of tasks and sub-tasks, where for each the teacher can set a different group of settings for the compiler to fit the theme of the exercise. Then, when after editing the teacher can publish the worksheet to the class and can monitor the progress for each student. The progress view of a student displays the state of the currently solved exercises, tasks with each the saved code of the last compilation. For this the update procedure can configured to be real time or to set intervals.

3.1 Language specification

A the basis of the work I took the language specification of Python[4] and based my implementation on it. I will implement the all the arithmetic, bitise or boolean operations. Further I will include the traditional flow constructs like the *if*, *while* and *for* statements. However it will be not a full implementation of this document. I have came to the decision to exclude some of the features described in it, because of complexity, time constraints or they had a mostly marginalised use case.

I have decided to not implement features like imaginary numbers, byte literals or the async and await keywords.

```
imagnumber ::=  (floatnumber | digitpart) ("j" | "J")
bytesliteral ::=  bytesprefix(shortbytes | longbytes)
bytesprefix  ::=  "b" | "B" | "br" | "Br" | "bR" | "BR" | "rb" | "rB" | "Rb" | "RB"
shortbytes   ::=  "'" shortbytesitem* "'" | '"' shortbytesitem* '"'
longbytes    ::=  "'''" longbytesitem* "'''" | '"""' longbytesitem* '"""'
```

[1]

And while I have lamented on the implementation of the generators and *yield expressions* as they are a higly useful feature of the language, I feel that the list or dictionary comprehensions are such a permanent aspect of Python that they have a higher priority.

Finally I will include an implementation of functions and classes, except the latter probably will have a much limited featureset.

[1]BNF notation of byte literal grammar description

3.2 Compiler

The transpiling of the Python code to JavaScript is done on the client side. From the dashboard the users can transpile and execute the written code. While the students are working on an exercises, the compiler will use the configurations set by the teacher for that given exercise.

From my research into parsers, and the Python language specification[4] I have decided to implement a LL(1) parser, as it is simple and limits the grammar rules to be sensible and easy to implement. It is also the type that the developers of Python are decided to use.

The compilation process consists of the creation of tokens from the stream of characters. Then the parser from the tokens in the lexical analysis creates an abstract syntax tree (AST). The nodes of this tree are the expressions and statements containing the references to the values and the variables. Given the run command the program is executed from the root node of this tree.

Before creating the tokens, the configuration file is read containing the set of rules for the compiler, to ignore. These rules will be used throughout the compilation process in all the phases, depending on the settings.

3.2.1 Rules

The structure of the file containing the compilation rules is of a *JSON*, where each rule consists of the name and its setting.

Listing 3.1: Example of the rules in the json file.

```
{
    "name" : "Name of the task",
    "description" : "Simple description of the exercise",
    "rules" : [
        "variables": {
            "names": { "allowed": "[a-z]{8}" },
            "globals": False,
            "locals": { "count": 3 }
        },
        "keywords" : { "forbidden" : ["def", "class", "for"] },
        ...
    ],
}
```

Variable limitations

- Variables can be limited by their name, from simply limiting it by the length, to writing a regular expression to for the disallowed values.
- Another setting can be the one of a black/white-list of usable variable types.
- Limiting the number of variables used in the given scope.
- With this setting one can disallow the use of global variables.

Keyword limitations

This rule consists of a map of keywords like *for*, *print* and their status.

Standard library limitations

With these rules the teacher can disable standard library functions like **sort**, or **math**.

Other limitations

- Further limitations include the ability to define functions.
- Or the capacity to allow or disallow the creation of classes.
- The capability to inherit from a class.

3.3 Interface

The client view consists of the student and the teacher page.

3.3.1 Student view

Dashboard This view is the main page the student sees after logging into the application. Here he can see the classes he is enrolled into, and the overview of the evaluation for each.

Exercises This is accessible after entering a class. Here are displayed all the exercises solved or yet to be completed. Each can be further expanded to show a description and the gained mark.

Editor This consists of a simple text editor, along with a interpreter and a result view.

3.3.2 Teacher view

Dashboard This view is an expansion of the students. Here the teacher can manage the classes he oversees.

Exercises On this page the teacher can create and assign exercises for the student groups, set the deadlines, point values, oversee each and every students solution. Further available are the statistics for view or export.

Editor One of the crucial part of the teachers view. Contains the same aspects the student has, but is extended with the ability to define the rules for the exercises. Also here are the test defined for the evaluation of the exercises.

Chapter 4

Implementation

4.1 Backend

4.1.1 Compiler

Todo

4.1.2 Database

Todo

4.2 User Interface

Todo

4.2.1 User - Student

Todo

4.2.2 User - Teacher

Todo

Chapter 5

Conclusion

5.1 Comparison

Bibliography

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools (2Nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[2] B. Alshaigy, S. Kamal, F. Mitchell, C. Martin, and A. Aldea. Pilet: An interactive learning tool to teach python. In *Proceedings of the Workshop in Primary and Secondary Computing Education*, WiPSCE '15, pages 76–79, New York, NY, USA, 2015. ACM.

[3] T. Fossum. Plcc: A programming language compiler compiler. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 561–566, New York, NY, USA, 2014. ACM.

[4] T. P. S. Foundation. Python language reference manual.

[5] J. H. Lasseter. The interpreter in an undergraduate compilers course. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 168–173, New York, NY, USA, 2015. ACM.

[6] PCMag-Encyclopedia. Definition of: compiler.

[7] P. Terry. *Compiling with C# and Java.* Pearson education. Pearson/Addison-Wesley, 2005.