PyVerDetector: A Chrome Extension Detecting the Python Version of Stack Overflow Code Snippets

Shiyu Yang[1], Tetsuya Kanda[1], Davide Pizzolotto[1], Daniel M. German[2], Yoshiki Higo[1]
[1]Graduate School of Information Science and Technology, Osaka University, Osaka, Japan
[2]Department of Computer Science, University of Victoria, Victoria, Canada
yangsy@ist.osaka-u.ac.jp, t-kanda@ist.osaka-u.ac.jp, davidepi@ist.osaka-u.ac.jp, dmg@uvic.ca, higo@ist.osaka-u.ac.jp

***Abstract*—Over the years, Stack Overflow (SO) has accumulated numerous code snippets, with developers going to SO for problem solutions and code references. However, in the case of the Python programming language, Python 3 is not necessarily backward compatible with Python 2. The major implication of this versioning problem is that code written in Python 2 may not be interpreted by Python 3 without modifications. This issue may affect the usability of Python code snippets on SO. We investigate how many Python code snippets on SO suffer from version compatibility issues, and find that about 10% of the snippets exhibit this problem. Moreover, of the code snippets that are interpretable only by Python 2 or Python 3, less than 17% are tagged with the Python version.**

In this paper, we present a Chrome extension called PyVerDetector. This extension allows the user to select a given version of Python and verifies whether the code snippets on a given SO question are compatible with the user's selected Python version, providing error messages if not. The tool parses snippets and can determine versioning errors due to differences in syntax and also provides the user with a list of Python versions capable of interpreting each code snippet.

***Index Terms*—Stack Overflow, Python version detector, Compatibility**

I. Introduction

Stack Overflow (SO) is a Q&A website for developers where users can post questions, answer questions, and search for content. The questions and answers on SO contain numerous code snippets, and this vast amount of ready-to-use code snippets provides developers with an easy way to find solutions to daily programming problems. Nowadays, copying code examples from SO is common [1].

While searching for the required code snippets on SO is convenient, recent studies have shown that code snippets can be toxic [2], obsolete [3], [4], and low-quality [5], leading to software quality issues [3], [6], license violations [7], or migration of security vulnerabilities [8].

There are many reasons why code snippets on SO are problematic. One of the reasons is the use of outdated programming language features in the code snippet. Existing programming languages constantly evolve to meet new needs. Popular programming languages often use versions to indicate their evolution, with more recent versions usually representing more mature forms of the language. Many popular programming languages are backward compatible, which means that programs compiled with an earlier language version can be compiled with a later version and exhibit the same behavior as the previous version. However, Python is known to break backward compatibility at almost every release (its backward compatibility policy, including its rules to break compatibility, is documented in PEP 387 [9]). Python 3.0 significantly broke backward compatibility with Python 2, while most releases make small changes that only affect a small proportion of features in the language (every release since Python 3.5 has removed deprecated features needed by older Python programs to run). The lack of backward compatibility of Python snippets may thus reduce their usability.

To understand how common SO Python code snippets have version compatibility issues, we conducted an empirical study. We structure our study by answering the following research questions:

- **RQ1: How many Python code snippets have version compatibility issues in the top answers to SO questions?**
  About 10% of code snippets have version compatibility issues in the top answers to questions.
- **RQ2: How many of the code snippets interpretable only by Python 2 or only by Python 3 are tagged with such Python version?**
  Only about 17% of code snippets interpretable only by Python 2 or only by Python 3 are tagged with the Python version.

Based on those results, we noticed that the version compatibility issues of Python code snippets on SO could not be ignored. When looking for a desired code snippet, users need to check whether the snippet can be interpreted by their desired Python version, and the tags of questions can not help users solve this issue.

In this work, we have developed a Chrome extension called **PyVerDetector** that detects whether a given code snippet on SO can be interpreted by the developer's selected Python version (2 or 3), and provides an error message if it cannot. This tool is publicly available on GitHub[1]. Although some research tools to detect Python versions have been developed, such as *PyComply* [10], these tools perform code parsing locally, which cannot provide real-time, online Python code snippets version checking for SO users.

In the rest of the paper, in order to save space, we will refer to a specific version of Python by its number only. For

[1]https://github.com/ysy-dlg/PyVerDetector

example, instead of Python 3.0 we represent it as 3.0.

II. Empirical study

To investigate the extent of code snippets with version compatibility issues on SO, we use Python interpreters to check Python code snippets from SO top answers. The reason for choosing top answers lies in the fact that they are expected to be correct answers and most likely to be used by other users.

A. Data Collection

To conduct the empirical study, we first need to obtain code snippets written in Python from the top answers posted on SO. We determine the top answer for each question based on the number of upvotes for the answer. We used the following two criteria to identify the code snippets required for this study:

- Code snippet from top answers with at least one question tag containing the word "Python".
- Posting data is the latest version.

We used SOTorrent [11], version SOTorrent20_03, to extract Python code snippets on SO. Based on our criteria above, we extracted 1,256,503 questions. Based on these questions we extracted 2,427,602 answers, of which 698,506 were top answers. Considering that an answer may contain multiple snippets, we extracted a total of 1,260,442 code snippets.

B. Code Snippet Analysis

We used multiple Python interpreters to parse Python code snippets; one major release of Python 2 (2.7) and four major releases of Python 3 (3.5 to 3.8). Other versions, older versions which are difficult to prepare an execution environment, and newer versions released after the dataset was released were excluded. A code snippet is considered to be interpretable by a Python version if it can be parsed by the Python interpreter for that Python version. This is checked by attempting to compile the script using the `py_compile` module. Otherwise, the code snippet is deemed uninterpretable by that Python version.

C. Results

1) RQ1: How many Python code snippets have version compatibility issues in the top answers to SO questions?

Analyzing the obtained Python interpreter parsing results of code snippets, we found that the code snippets can be divided into the following five categories:

- **Pass all versions (Pass all):** The code snippet passes parsing for Python 2 and all Python 3 versions.
- **Fail for all versions (Fail for all):** The code snippet fails for parsing for Python 2 and any Python 3 versions.
- **Pass Python 2, Fail for some Python 3 (Pass 2&3):** The code snippet passes parsing for Python 2 and at least one parsing for Python 3, but not all Python 3 versions.
- **Fail for Python 2, Pass all or some Python 3 (Only pass 3):** The code snippet fails for Python 2 parsing but passes all or some Python 3 parsing.
- **Fail for all Python 3, Pass Python 2 (Only pass 2):** The code snippet fails for any Python 3 parsing but passes Python 2 parsing.

TABLE I: Compatibility Of Code Snippets In Top Answer.

Categories	#Snippets	Percentage		
Pass all	760,394	60.33%		
Fail for all	382,275	30.33%		
Pass 2&3	131	0.01%	**Tagged version**	**Percentage**
Only pass 3	48,045	3.81%	10,157	21.14%
Only pass 2	69,597	5.52%	9,122	13.11%

Of the above categories, the results are shown in the first three columns of Table I. As shown in the table, "Pass 2&3", "Only pass 3", and "Only pass 2" are the three types of code snippets with version compatibility issues.

In addition, the data shows that nearly 30% of the code snippets are not executable by any Python version in this study. This means that these code snippets fail for all Python versions. There are several possible reasons: (1) Code snippets using old Python versions such as 2.0 and 3.0. (2) Answers tagged "Python" containing non-Python code snippets, such as program output and code snippets written in other languages. (3) Programming errors, such as syntax errors.

Answer to RQ1: About 10% of code snippets have version compatibility issues in the top answers to questions.

2) RQ2: How many of the code snippets interpretable only by Python 2 or only by Python 3 are tagged with such Python version?

Questions on SO require tags to be assigned to describe the topic of the question. "Python 2.x" and "Python 3.x" tags now exist on SO for users to tag questions for Python 2 or Python 3 only. For example, if the "Python 3.x" tags correctly identify a code snippet that cannot be used by Python 2, then users can avoid using that code snippet in Python 2. We would like to know how many of the code snippets that are interpretable only by Python 2 or interpretable only by Python 3 are tagged with such Python version.

For this purpose, we further processed the parsing results of the code snippets. The results are shown in Table I. The fourth row of the table is a code snippet that can only be interpreted by Python 3. Its fourth column corresponds to the number of code snippets that have the tag "Python 3.x" in the question. The last row of the table is for code snippets that can only be interpreted by Python 2. The fourth column corresponds to the number of code snippets with the "Python 2.x" tag in the question.

Answer to RQ2: Only about 17% of code snippets interpretable only by Python 2 or only by Python 3 are tagged with the Python version.

It is clear that the version compatibility issues of code snippets exist at SO and cannot be ignored. However, since code snippets are not well tagged with the Python version, SO users cannot simply determine the Python version of the snippet by the tag, which may lead to misuse of the snippet. Therefore, we need to provide a tool for users to determine the Python version of Python code snippets on SO.

III. PyVerDetector

We developed a Chrome extension, PyVerDetector to help SO users address the issue of version compatibility of Python

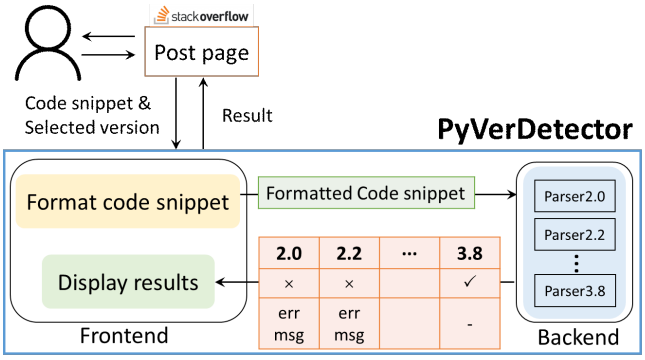

Fig. 1: Overview of PyVerDetector.

code snippets. PyVerDetector has two main features:

1) For each code snippet inside a code block, the tool determines if the code snippet is compiled without errors using the user-selected Python version.
2) If not, the tool provides the error message and the location of the error.

PyVerDetector consists of two components: a frontend part (running on the user's browser) responsible for fetching the code snippet from SO and displaying the results, and a backend part (running on a server) responsible for statically analyzing the given code snippet across multiple Python versions. The overview of PyVerDetector is shown in Figure 1. Upon loading a SO page, for each Python snippet, the frontend calls the backend and retrieves the parse result containing all the supported Python versions for that snippet. Finally, the frontend alters the page to present the result to the user. PyVerDetector supports versions (2.0 to 2.7, 3.0 to 3.8).

A. Frontend

The frontend has two main features:

1) Format code snippet: The frontend fetches Python code snippets from the page, formats the code snippets of the REPL mode, and sends them to the backend for parsing. The frontend also copies the formatted code snippet to the clipboard for the user to use.

2) Display result: The result from the backend contains the parsing results of the code snippet for all Python versions. The frontend presents them to the user according to the Python version selected by the user in the drop-down menu inserted below the code snippet. The frontend shows the following two types of messages on the page:

- The parsing result of the user-selected Python version (default: 3.8). If the code snippet passes the parsing of that version, the message "No error for Python X.X" is displayed. Otherwise, the error message and the line number of the error that occurred are displayed.
- If there are some Python versions other than the selected one which passes the parsing of the code snippet, output them like "Also works for: Python 3.8".

B. Backend

The backend part is inspired by CPython and PyComply [10]. Since it is not practical to wrap multiple execution environments of the old Python versions inside a Chrome extension, we decided to use an Abstract Syntax Tree (AST) parser following the grammar of Python. If the Python snippet can be parsed, with a grammar for a specific Python version, we assume the snippet is compliant for the particular version. We used a combination of *Flex*[2] and *Bison*[3] to generate the code snippet's AST, reporting compliance only in case this AST is generated successfully.

Naively wrapping the output of *Flex* and *Bison*, however, is not sufficient. Unlike PyComply, the web-based nature of our tool required us to perform heavy modifications to the generated parsers in order to support asynchronous invocations, and join multiple parsers together in a single executable. In our approach, a code snippet is tested against each grammar sequentially, and the various error message collected in a JSON message to be sent to the frontend.

Finally, in order to run the backend inside a web extension, we used the *Emscripten* toolchain[4] to compile the original C code into cross-platform WebAssembly to be bundled inside the extension.

C. Python Grammars and Extensibility

Being our tool based on the *Flex* and *Bison* parser generators, it implies the necessity of having an input grammar representing the Python Language. While writing a different grammar for each Python version is certainly not impossible, being up-to-date with the annual Python release schedule by manually writing a new grammar for each release would require considerable effort nonetheless. Fortunately, the Python website provides the full changelog[5] and grammar of each released version since Python 2.2[6].

These grammars, however, are written for a LL(1) parser, while *Bison* is an LALR(1) parser. Despite every LL(1) grammar being LR(1), but not necessarily LALR(1) [12], these two in practice have a great intersection. For this reason, we managed to write a tool to convert the provided grammars from LL(1) EBNF syntax found in the Python archives to LALR(1) Bison syntax expected by our parser generator.

Unfortunately, since version 3.10, CPython switched from a LL(1) parser to a PEG parser [13], with the grammars being provided only in PEG syntax since Python 3.9. Converting from a PEG grammar to LALR(1) is not as easy as converting from a LL(1) to LALR(1). In fact, the equivalence between PEG and Context-Free Grammars such as EBNF has been proven undecidable [14]. For this reason, our extension works with Python versions up to 3.8, but to extend it to future versions of Python, an additional parser for PEGs should be wrapped alongside the current Context-Free Grammar parser.

[2]https://github.com/westes/flex
[3]https://www.gnu.org/software/bison
[4]https://emscripten.org/
[5]https://docs.python.org/3/whatsnew/changelog.html
[6]https://docs.python.org/release/2.2/ref/grammar.txt

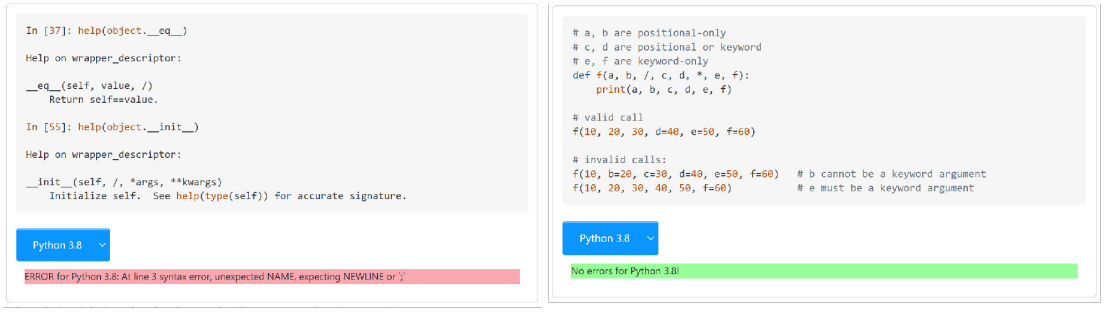

Fig. 2: Example of how the extension works in the default version. (Left): Parse error, (Right): Pass.

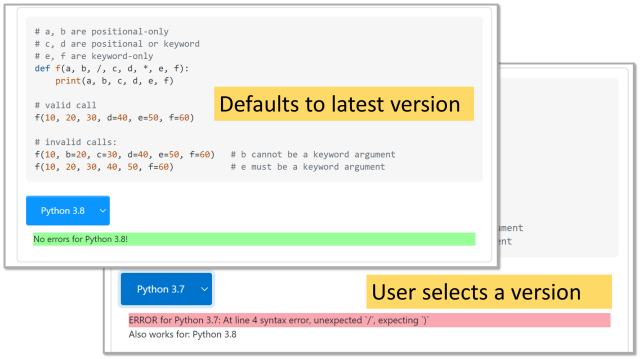

Fig. 3: User manually selects a version of Python, incompatible with the given snippet.

IV. Usage Scenarios

A. Displaying the latest version as default

The default value displayed by PyVerDetector is the parsing result for the latest available version. As shown in Figure 2, when the user opens a SO page[7] with some Python code snippets, PyVerDetector will immediately display its parsing results for 3.8 for all Python code snippets on the page. We use green to show pass messages and red to show error messages. This allows the user to quickly get an at-a-glance view of the compatibility of the code snippets for a recent Python version.

B. Accurate display of results for selected versions

When the user wants to know the compatibility of a code snippet for a particular Python version, PyVerDetector can show the user exactly the relevant information. As shown in Figure 3, the same question in Figure 2, the user has selected 3.7, and PyVerDetector returns an error message that the code snippet cannot be interpreted by 3.7 because the "Positional-only parameters" is used in the fourth line of the snippet. This feature, in fact, has only been supported since 3.8.

V. Evaluation of Accuracy

In this section, we evaluate the accuracy of PyVerDetector and compare it with an existing tool: PyComply [10].

We apply PyVerDetector and PyComply to the dataset of top answer code snippets and obtain their respective parsing results using the same method as described in Section II-A. We use the parsing results of Python interpreters as the ground truth. The accuracy of PyVerDetector and PyComply was measured by comparing their respective parsing results with the ground truth in terms of code snippets. We evaluate PyVerDetector using the well-known metrics for binary classification: precision, recall, and accuracy.

[7]https://stackoverflow.com/questions/28243832

TABLE II: Comparison results of PyVerDetector (PyVer) and PyComply (PyC)

Python Version	Precision		Recall		Accuracy	
	PyVer	PyC	PyVer	PyC	PyVer	PyC
Ver2.7	98.28%	98.28%	99.95%	99.95%	98.82%	98.82%
Ver3.5	98.02%	98.02%	99.98%	99.98%	98.72%	98.72%
Ver3.6	97.98%	97.98%	99.98%	99.98%	98.67%	98.67%
Ver3.7	97.98%	-	100.00%	-	98.68%	-
Ver3.8	97.98%	-	100.00%	-	98.67%	-

The results are shown in Table II. We can see that PyVerDetector (PyVer) and PyComply (PyC) have the same accuracy for the three Python versions 2.7, 3.5, and 3.6. However, PyVerDetector can provide code detection for the two newer Python versions, 3.7 and 3.8, and has shown high accuracy in both versions.

VI. Conclusion

In this paper, we conducted an empirical study to understand the extent of Python code snippets in SO that have version compatibility issues. We found that version compatibility issues exist in SO code snippets. In response, we developed a Chrome extension, PyVerDetector, which can identify versioning errors due to different syntax. PyVerDetector helps users detect whether the code snippets on a given SO question are compatible with their selected Python version and provides error messages if not. We evaluated PyVerDetector by comparing it with PyComply, showing comparable performance but with newer versions supported by our tool, making code snippet detection on SO more convenient and efficient.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Numbers JP20H04166, JP21K18302, JP21K11820, JP21H04877, JP22H03567, JP22K11985, JP19K20239.

REFERENCES

[1] S. Baltes and S. Diehl, "Usage and attribution of stack overflow code snippets in github projects," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1259–1295, 2019.

[2] C. Ragkhitwetsagul, J. Krinke, M. Paixao, G. Bianco, and R. Oliveto, "Toxic code snippets on stack overflow," *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 560–581, 2019.

[3] H. Zhang, S. Wang, T.-H. Chen, Y. Zou, and A. E. Hassan, "An empirical study of obsolete answers on stack overflow," *IEEE Transactions on Software Engineering*, vol. 47, no. 4, pp. 850–862, 2019.

[4] J. Zhou and R. J. Walker, "Api deprecation: a retrospective analysis and detection method for code examples on the web," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 266–277.

[5] Y. Wu, S. Wang, C.-P. Bezemer, and K. Inoue, "How do developers utilize source code from stack overflow?" *Empirical Software Engineering*, vol. 24, no. 2, pp. 637–673, 2019.

[6] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack overflow considered harmful? the impact of copy&paste on android application security," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 121–136.

[7] L. An, O. Mlouki, F. Khomh, and G. Antoniol, "Stack overflow: A code laundering platform?" in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 283–293.

[8] M. Verdi, A. Sami, J. Akhondali, F. Khomh, G. Uddin, and A. K. Motlagh, "An empirical study of c++ vulnerabilities in crowd-sourced code examples," *IEEE Transactions on Software Engineering*, 2020.

[9] B. Peterson and B. Cannon, "Backwards compatibility policy," PEP 387, 2009. [Online]. Available: https://peps.python.org/pep-0617/

[10] B. A. Malloy and J. F. Power, "Quantifying the transition from python 2 to 3: An empirical study of python applications," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2017, pp. 314–323.

[11] S. Baltes, C. Treude, and S. Diehl, "Sotorrent: Studying the origin, evolution, and usage of stack overflow code snippets," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 2019, pp. 191–194.

[12] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: principles, techniques, & tools*, 2nd ed. Pearson, 2007, p. 242.

[13] G. van Rossum, P. Galindo, and L. Nikolaou, "New peg parser for cpython," PEP 617, 2020. [Online]. Available: https://peps.python.org/pep-0617/

[14] B. Ford, "Parsing expression grammars: a recognition-based syntactic foundation," in *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2004, pp. 111–122.