Checking Conformance to a Subset of the Python Language

Michel Wermelinger
School of Computing and Communications
The Open University
Milton Keynes, United Kingdom
michel.wermelinger@open.ac.uk

ABSTRACT

Introductory courses usually only teach a small subset of a programming language and its library, in order to focus on the general concepts rather than overwhelm students with the syntactic, semantic and API minutiae of a particular language.

This paper presents courseware that checks if a program only uses the subset of the Python language and library defined by the instructor. This allows to automatically check that programming examples, exercises and assessments only use the taught constructs. It also helps detect student code with advanced constructs, possibly copied from Q&A sites or generated by large language models.

The tool is easy to install, configure and use. It also checks Python code in Jupyter notebooks, a popular format for interactive textbooks and assessment handouts.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; **Computational science and engineering education**; **Student assessment**; • **Software and its engineering** → Automated static analysis; • **Applied computing** → Computer-managed instruction.

KEYWORDS

code checking, programming exercises, novice programming, introductory programming, academic integrity

ACM Reference Format:
Michel Wermelinger. 2023. Checking Conformance to a Subset of the Python Language. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 2 (ITiCSE 2023), July 8–12, 2023, Turku, Finland*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3587103.3594155

1 INTRODUCTION

While languages like Scratch have been designed with novices in mind, most higher education courses use industrial languages [1]. Courses often only use a small subset of a programming language and its library, in order to reduce cognitive load, to focus on concepts rather than linguistic details, to ease marking, to set a common level playing field for all students, and to prevent plagiarism. In data structures and algorithms (DSA) courses, avoiding advanced language features and built-in functions with 'hidden' loops and data structures helps make the complexity of algorithms more explicit.

It can easily happen that students and instructors deviate from the taught subset, e.g. when generating code with AI tools [2], but it is hard and tedious to detect such deviations, especially in courses with large cohorts and thousands of lines of code in the teaching and assessment materials.

This paper presents `allowed`, a tool that takes as input Python code and a definition of a subset of the Python language and library, and checks if the code conforms to that subset. The tool can be applied to any course (CS0–2, data science, bioinformatics, computational engineering, etc.) that uses a restricted subset of Python because the subset is defined by the instructor.

The tool is available[1] under an open source licence that allows redistribution and modification. The tool is implemented in pure Python and runs on Linux, macOS and Windows. The rest of this paper explains how the tool can be used.

2 USE CASES

`allowed` supports course design, preparation and presentation. The tool assumes that a course is organised in 'units' (lessons, weeks, textbook chapters, whatever is appropriate) and that Python constructs introduced in one unit are also allowed in subsequent units. The tool requires instructors to list the taught constructs, unit by unit, in one short file (see Section 4).

Thus, the tool's configuration becomes a reference document that makes the course's language content and progression explicit. This may facilitate the onboarding of new tutors and the replacement of instructors, e.g. during a sabbatical. More importantly, it helps educators discuss the design of the course: Are important constructs missing? Are the pre-requisites of subsequent courses met? Are too many constructs being taught? If so, which units are overloaded? Is the pace of introducing new constructs too fast? Is the progression of constructs appropriate?

During course preparation, instructors can use the tool to check that the teaching and assessment materials only use the taught constructs, to avoid students getting confused. Due to their experience with Python, instructors may inadvertently use advanced constructs and `allowed` can catch such mistakes.

Finally, during course presentation, students and tutors can check code before and after it is submitted for formative or summative assessment. Even though `allowed` can be part of an auto-grading pipeline, it is best used to complement and support human marking. Tutor feedback based on the reported deviations from the taught

ITiCSE 2023, July 8–12, 2023, Turku, Finland

ACM ISBN 979-8-4007-0139-9/23/07.
https://doi.org/10.1145/3587103.3594155

[1] https://dsa-ou.github.io/allowed

constructs can help students think about alternative solutions that may be simpler, more efficient, and more readable.

3 INSTALLATION AND USAGE

The tool is made of two files: the program and a configuration file that defines the Python subset. By default, the tool uses the configuration for our DSA course. The tool is run from the command line. Typing

```
python allowed.py path/to/folder
```

checks all Python files and Jupyter notebooks in the given folder and its subfolders against the provided default configuration. This allows students to check multi-file projects, tutors to check several students' submissions, and instructors to check all course materials in one go. The tool can also be run on individual files, for example by typing

```
python allowed.py file1.py file2.ipynb
```

The tool prints a list of disallowed constructs and where they occur. For example,

```
file1.py:45: else in while-loop
file2.ipynb:cell_3:1: else in while-loop
```

indicates that both the code file and the notebook have an `else` branch in a `while` loop, which is not taught in our course.

By default, the tool checks code against *all* the allowed constructs, but with command line option `-u N` it checks code only against those constructs introduced up to and including unit `N`. In this way, it is possible to make sure that each learning and teaching resource (mid-term test, lab exercise with starter code, etc.) only draws on the constructs taught until the resource's unit.

Like other static analysis tools, `allowed` processes an abstract syntax tree and thus requires the input to be syntactically valid Python code: if the code cannot be parsed, it cannot be checked. The tool does not process Python files that have syntax errors, but for Jupyter notebooks, it skips the code cells that contain syntax errors and processes all others, in order to check as much code as possible.

The allowed Python subset may include only some methods, e.g. `append()` but not other list methods like `count()`. To check if `variable.method()` is allowed, the tool needs to know the type of `variable`, which is obtained with the `pytype` type checker[2], if it is installed. Type checking slows down the code analysis, so method call checks must be explicitly enabled with a command line option.

Our DSA course comprises 1,075 Jupyter notebooks and 13 Python files. The large number of notebooks is mostly due to each exercise's hint and answer being in its own notebook. In total, the notebooks and code files have almost 5,700 lines of code, not counting blank lines and comments. Running `allowed` on all files takes about 103 seconds with method call checks and 3 seconds without, on an iMac with an M1 chip and 16 Gb RAM. The tool does not make use of multiple cores.

4 CONFIGURATION

The tool comes with the JSON configuration file for our DSA course. Here is an excerpt:

```
{
  "LANGUAGE": {
    "2": ["=", "name", "def", "min", ...],
    "3": ["if", "and", "or", "not", "==", "<", ...],
    "4": ["for", "while", "list literal", ...],
    ...
  },
  "IMPORTS": {
    "2": {"math": ["floor", "ceil", "trunc", "pi"]},
    "7": {"collections": ["deque"]},
    "17":{"typing": ["Hashable"], "random": ["random"]},
    ...
  },
  "METHODS": {
    "4": {"List": ["insert", "append", "pop", "sort"]},
    "7": {"collections.deque": ["append", "pop", ...]},
    "11": {"Set": ["pop"]},
    ...
  }
}
```

`"LANGUAGE"` maps each unit to the Python constructs, operators and built-in functions introduced in that unit. In our course, unit 2 introduces assignments, names (identifiers), function definitions, function `min()`; unit 3 introduces Booleans and related constructs; unit 4 introduces loops and sequence types; and so on. Most constructs are indicated by the corresponding keyword (like `def`), but others, like list literals, must be explicitly described. The tool checks that all strings correspond to existing Python constructs.

`"IMPORTS"` maps each unit to the modules and the list of exported objects introduced in that unit. In our course, unit 2 only allows four of the functions and objects of module `math`. The tool checks for inconsistencies between `"LANGUAGE"` and `"IMPORTS"`, like allowing to import modules before the `import` statement is introduced.

`"METHODS"` maps each unit to the classes and methods introduced in that unit. For example, unit 11 only introduces one method on sets. The input code is checked against the indicated methods only if `pytype` is installed and the methods call option is given in the command line.

If a course is not divided into units, then `"LANGUAGE"`, `"IMPORTS"` and `"METHODS"` must list all constructs under the same unit, e.g. `"1"`.

As can be seen, the configuration format is straightforward and the provided configuration file can be easily modified in any text editor to suit the needs of other courses. The tool supports a command line option to select which configuration file to use. This allows instructors and students to keep configurations for all their courses and switch between them.

[2] https://google.github.io/pytype

ACKNOWLEDGMENTS

I thank Tony Hirst for ideas to improve the tool, tutor Andy Connolly for testing the tool on his students' assignments, and students Michael Snowden and Jake Robson for implementing some features.

REFERENCES

[1] Simon, Raina Mason, Tom Crick, James H. Davenport, and Ellen Murphy. 2018. Language Choice in Introductory Programming Courses at Australasian and UK Universities. In *Proc. 49th Technical Symposium on Computer Science Education*. ACM, 852–857. https://doi.org/10.1145/3159450.3159547

[2] Michel Wermelinger. 2023. Using GitHub Copilot to Solve Simple Programming Problems. In *Proc. 54th Technical Symposium on Computer Science Education*, Vol. 1. ACM, 172–178. https://dl.acm.org/doi/10.1145/3545945.3569830