PyToPseu: Automatic Natural-Language Formulations of Programming Constructs to Avoid Misconceptions

hep/

Jean-Philippe Pellet and Patrick Wang

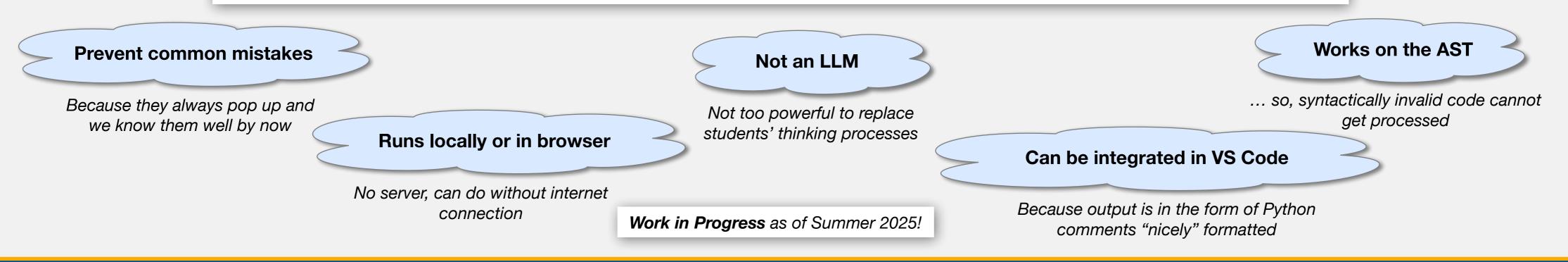
University of Teacher Education, Lausanne, Switzerland jean-philippe.pellet@hepl.ch(≥), patrick.wang@hepl.ch

ISSEP 2025 September 8–11 Trier, Germany

Introduction & Context

Introduction to programming remains one of the first delicate topics in computer science education. We present a tool which, from Python code, generates a line-by-line natural-language description of the code.

Its intended use is first and foremost a reading and interpretation tool for existing code, but it can be used to verify the meaning of code being written as well.



Example Outputs



Functions

```
def add_abs(x, y) \rightarrow int:
                                       define the function add abs, which accepts 2 arguments,
                                               \perp <u>x</u> and y, and which returns an integer number, like so:
                                            get out of the function returning the sum of (the absolute value of \underline{x}) and
    return abs(x) + abs(y)
                                                   └ (the absolute value of y)
def b(x, y, z) \rightarrow int:
                                       define the function \underline{b}, which accepts 3 arguments, \underline{x}, y and
                                               \perp <u>z</u>, and which returns an integer number, like so:
                                            get out of the function returning the sum of x and 1
    return x + 1
                                          Spell out function, arguments, return type
                                          Insist that return terminates the function
```

```
in <u>a</u>, store an anonymous function without any arguments
a = lambda: 42
                                                 └ which returns 42
                                         in \underline{a}, store an anonymous function which accepts one argument, \underline{x}, and
a = lambda x: x ** 4
                                                 ^{ldash} which returns \underline{x} to the power of 4
                                               Lambda functions are also supported
                                                 (albeit without type annotations)
```



- Usefulness tests in classrooms
- Iteration with students on best phrasing
- More precise phrasing with type info from mypy/pylint

[3] Introduction of the block model and the various levels of understanding of a program

• If useful: better technical integration with IDEs

[4] Schulte, C., Clear, T., Taherkhani, A., Busjahn, T., Paterson, J.H.: An introduction to program comprehension for computer science educators. In: Proceedings of the 2010 ITiCSE working group reports pp. 65–86 (2010)

Relevance of reference in the context of this poster:

[4] Comparison of models for program comprehension

^[1] Chiodini, L., Moreno Santos, I., Gallidabino, A., Tafliovich, A., Santos, A.L., Hauswirth, M.: A curated inventory of programming language misconceptions. In: Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1. pp. 380–386 (2021)

^[2] Qian, Y., Lehman, J.: Students' misconceptions and other difficulties in introductory programming: A literature review. In: ACM Transactions on Computing Education (TOCE) 18(1), 1–24 (2017) [3] Schulte, C.: Block model: an educational model of program comprehension as a tool for a scholarly approach to teaching. In: Proceedings of the fourth international workshop on computing education research. pp. 149–160 (2008)

^[1] List of misconceptions classified by programming language [2] Literature review of misconceptions