УДК 004.432

Student D.A. Astrouski
Scientific supervisor Senior Lecturer E.V. Kryvonosova
(Department Intercultural Communication and Technical Translation, BSTU)

PYTHON INTERPRETERS AND WHY MYPY IS OUR FUTURE?

High-level programming languages work inside much more complicated than it seems to be. High-level programming languages are popular in the market today because everyone can start working on them much faster than using some assembly code or c, java and others, and the popular conclusion is that they are just slower and they have less instruments for developers. There is some truth inside, but not everything is that easy and there are many reasons why they are still relevant for market purposes.

Translators are used to translate our readable code into machine one. There are two types of them: compiler and interpreter. Compiler reads all the code we have and converts it to machine code at once, and the interpreter is doing the same, but it reads the code line-by-line in real time. Most high-level languages like JavaScript, Python and Ruby are using interpreters. If we have a large application with a little bug on 2538's line of code in one of 400 libraries, the interpreter solves this problem by editing each line of code without recompiling. Interpreters are more convenient for large cross-platform applications because they do not need to rewrite code for each hardware architecture.

Why do low-level languages for some fast applications do not use interpreters? Popular misconception is that this technology was not as popular during the times when programming was just coming to 'public'. Low-level languages need to interact with hardware directly; interpreting abstractions will only reduce application speed. All the programming languages can be both compiled and interpreted, they can use all the features like dynamic debug that we can use in stock Visual Studio and improve our developer performance.

Now we can analyse interpreters using the popular programming language Python with dynamic typing and its most popular interpreters: MyPy, PyPy, CPython. All of them have the same technologies inside: they have 3-layer memory management, global interpreter lock for multithreading, magic methods support, object-oriented programming, but they have different speed of work that can cause a lot of questions and most of the answers can be obvious. Most of the interpreters are written in other languages, like C for CPython or C# for IronPython. They can create some additional features. Python is a dynamic typing programming language, but it can be a static typed programming language without changing the language itself; it happened with JavaScript and TypeScript.

If we consider typing, dynamic typing is the advantage and disadvantage at the same time for the developer and for the interpreter. Programmers and computers cannot know exactly what type of data will be input in the next line of code, and it needs a lot of checks under the hood that are slowing the system down. For some newbies, it would be great not to waste their time thinking of what type to use, and nothing requires keeping in mind what type of variable will be there.

We can use DocStings and mark it for IDE, but anyway it will not be as easy as static types, and we cannot specify variables that are announced in the code. DocString is a comment for developers, but the interpreter usually ignores it.

In 2015, Python 3.5 was released, and the most discussed feature was type annotation, which allows specifying the type for each variable on the IDE level (but not the interpreter), making it easy for linter to mark all the necessary data. Using this feature, dynamic typing language cannot work with speed as a static typed variable, but MyPy fixed it. All interpreters are using static typing under the hood all the time, but the way they realized what type is used for a variable was never as easy as with type annotations with MyPy.

Typing is not the only thing that is used to optimize the code. A PyPy counts as a faster interpreter because it uses JIT (Just-in-Time) compilation. But MyPy has a lot of other features, such as custom Garbage Collection (used to clear memory), custom PyCache (used to keep bytecodes that are not changed and frequently used), and with all these features the interpreter can work as fast as C and other low-levels in most cases keeping all the interpreter features.

There are still some features that could be hard to implement without low-level languages; but with these features, there are no reasons to use low-level languages for all the scripts that we want to write.

REFERENCES

1. R. Nystrom. Crafting Interpreters: A Handbook for Making Programming Languages. Genever Benning, 2021.

2. A. Ranta, Implementing Programming Languages. An Introduction to Compilers and Interpreters College Publications, London, 2012.

3. M. L. Scott. Programming Language Pragmatics. Morgan Kauffman, 2006.