Optional Type Annotations for Python

Paul Prescod

ISOGEN Consulting Engineer
paul@prescod.net

The primary difference between Python and a language like Java is that Python does not require type declarations. This allows Python programmers to write wonderfully elegant dynamic code. Most Python users would agree that this is a great source of Python's power.

And yet, this dynamicity comes with a high price. The techniques necessary to optimize a language as dynamic as Python seem to be out of reach of the resources of the Python community, if they exist at all. Type declarations in most other languages help the compiler and/or interpreter to improve the performance of the compiler. Undergraduate students routinely write compilers for strongly typed languages and these compilers generate reasonably efficient assembly language.

The other major cost of dynamicity is compile time safety. Finding type errors or attribute errors in Python can sometimes be much harder than in languages where types and attributes are frozen in advance. It is undeniably true that type checks do not substitute for proper testing but it is also true that finding errors early in the development process is cheaper than finding them later. A static type checking system can find some kinds of errors earlier in the process.

The members of the Types-SIG are working on a mechanism for annotating Python code with type declarations. The details of the proposal are still under development but some of the basic principles are already becoming visible.

Adding a major feature like static type checking to Python is an extremely difficult project because every effort must be taken not to interfere with Python's existing features. In particular, a static type system must not prevent or punish dynamicity. It must also not be "contagious": programmers shouldn't need to add type annotations to their code because they use a module that uses type annotations. Conversely it should be easy for annotated code to call into ordinary Python.

An optional type annotation system should also acknowledge that Python has advanced support for generic (not genetic!) data structures and algorithms. For instance a Python sort algorithm can work on any type of writeable sequence containing elements of any type. Therefore a type annotation system should also support generic data structures and algorithms.

Finally an annotation system must be related to an interface declaration system. Python functions and methods should work with objects with a certain interface or protocol, not on subtypes of some pre-declared type.

This poster describes the current proposal by the members of the Types-SIG. Hopefully we will be able to get wide feedback from members of the Python community.