PEP 484

Type Hints

- Dmytro Ryzhkov
- John Murphy
- Joseph Vostrejs
- Bu Sun Kim

PEP 484 – Type Hints

- created September 2014
- accepted for Python 3.5
- established standard syntax for type annotations

PEP 484 – Type Hints

- Python is *dynamically typed*
- however, **function annotations** have existed since Python 3.0 (2006)

Goals

- allow easier static analysis and refactoring of Python code
- provide standard notation for use by IDEs for code completion and refactoring
- create potential for runtime type checking by third party packages

Non-Goals

- implementing runtime type checking *within* Python
- "It should also be emphasized that Python will remain a *dynamically typed language, and the authors have no desire to ever make type hints mandatory, even by convention.*"

Function Annotations

```
def greeting(name):
    return 'Hello ' + name

def greeting_annotated(name: str) -> str:
    return 'Hello ' + name
```

Function Annotations

- no type checking occurs at runtime based on function annotations
- function annotations were put in place with the goal of using them for type checking in the future

Type Hints Prior to PEP 484

- mypy (eventually implemented for PEP 484)
- Reticulated Python (slightly different from mypy)
- pyflakes, pylint, pychecker
- PyCharm by JetBrains

Regular Python

```
def fib(n):
    a, b = 0, 1
    while a < n:
        yield a
        a, b = b, a+b
```

Statically Typed Python

```
def fib(n: int) -> Iterator[int]:
    a, b = 0, 1
    while a < n:
        yield a
        a, b = b, a+b
```

Acceptable Type Hints

- built-in classes
- abstract base classes
- types available in the types module
- user-defined classes

Rejected Alternatives for Type Annotation Syntax

Angular brackets (e.g. List<int>) as in languages like C++, Java, C# and Swift

- desire for type expressions to be syntactically the same as other expressions
- '<', '>' tends to be harder to parse
- languages that use it either have to utilize powerful backtracking techniques or only allow it in specific contexts to prevent ambiguity

Parsing Ambiguity for Angular Brackets

```
a < b > [ c ]

(a<b>)[c]          # I.e., (a<b>).__getitem__(c)
a < b > ([c])      # I.e., (a < b) and (b > [c])
```

Rejected Alternatives for Type Annotation Syntax

the double colon (::)

- it's ugly, and reduces readability
- used for other purposes in other languages (like scoping in C++)
- because it's new syntax it would only work with python 3.5 code, existing syntax allows the current proposal to work for older versions of Python 3

Potential Issues with the Accepted for Syntax Type Annotation

- no direct incompatibilities are introduced, however, the goal is to eventually use annotations only for type hints
- there are existing uses for function annotations that are incompatible with type hinting, and may confuse a static type checker and cause it to send out extra warnings or errors.

Problems with Forward Declarations

- Python requires all names to be defined by the time they are used.
- annotations are evaluated at the time a function is defined, so any names used in an annotation must be already defined when the function is being defined
- a common scenario is a class definition whose methods need to reference the class itself in their annotations.

Problems with Forward Declarations

```
class Node:
  """Singly Linked List"""

  def __init__(self, next: Node):
      self.next = next
```

Problems with Forward Declarations

- code shown on previous slide will not work as written
- current solution is to allow use of string literals in annotations
- most of the time should not be an issue

Behavior of Type Checkers

- functions without annotations should be treated as having the most general type possible or ignored by type checkers
- type checkers should check the body of a checked function for consistency with the annotations given
- they can also check correctness of calls appearing in other checked functions

Sources

- https://www.python.org/dev/peps/pep-0484/
- https://www.python.org/dev/peps/pep-0482/
- http://mypy-lang.org/
- Vitousek, Michael M., Andrew M. Kent, Jeremy G. Siek, and Jim Baker. "Design and Evaluation of Gradual Typing for Python." ACM SIGPLAN Notices SIGPLAN Not. 50.2 (2014): 45-56. Web.