C A brief introduction to type hints

Python introduced type hints (or "type annotations") as an official part of the language through PEP 484 and Python version 3.5. Since then, type hints have become more common throughout many Python codebases and the language has added more robust support for them. Type hints are used in every source code listing in this book. In this short appendix, we aim to introduce type hints, explain why they are useful, explain some of their problems, and link you to more in-depth resources about them.

Warning

This appendix is not meant to be comprehensive. Instead, it's a brief kickstart. Please see the official Python documentation for details. https://docs.python.org/3/library/typing.html

C.1 What are type hints?

Type hints are a way of annotating the expected types of variables, function parameters, and function return types in Python. In other words, they are a way that a programmer can indicate the type that is expected in a certain part of a Python program. Most Python programs are written without type hints. In fact, before reading this book, even if you are an intermediate Python programmer, it is very possible that you have never seen a Python program with type hints.

Because Python does not require the programmer to specify the type of a variable, the only way to figure out the type of a variable without type hints is through inspection (literally reading the source code up to that point or running it and printing the type) or documentation. This is problematic because it makes Python code harder to read (although some would say the opposite, and we will get to that later in this appendix). Another issue, is that because Python is very flexible, it allows the programmer to use the same variable to refer to multiple types of objects, which can lead to errors. Type hints can help prevent this style of programming and alleviate these errors.

Now that Python has type hints, we call it a "gradually typed" language—meaning you can use type annotations when you want to, but they are not required. In this short introduction I hope to convince you (despite perhaps your resistance to how they fundamentally change the look of the language) that having type hints available is a good thing. A good thing that you should take advantage of in your code.

C.1.1 What do type hints look like?

Type hints are added to the line of code where a variable or a function is declared. A colon (`:`) is used to indicate the start of a type hint for a variable or a function parameter, and an arrow (`->`) is used to indicate the start of a type hint for a function return type. For example, take the following line of Python code:

```
1

def repeat(item, times):
```

copy

Without reading the function definition, can you tell what this function is supposed to do? Is it supposed to print out a string a certain number of times? Is it supposed to do something else? Of course, one could read the function definition to figure out what it's supposed to do, but that would take more time. The author of this function has also, unfortunately, not provided any documentation. Let's try it again with type hints.

```
def repeat(item: Any, times: int) -> List[Any]:
```

copy

That's a lot clearer. Just looking at the type hints, it would appear this function takes an `item` of `Any` type and returns a `List` filled with `times` number of that item. Of course, documentation would still help make this function more understandable, but at least the user of this library now knows what kind of values to supply it and what kind of value it can be expected to return.

Suppose the library that this function is supposed to be used with, only works with floating point numbers and this function was meant to be used as a setup for lists to be used in other functions. Then, we can easily change the type hints to indicate the floating point constraint.

```
1

def repeat(item: float, times: int) -> List[float]:
```

copy

Now, it's clear that `item` must be a `float` and the returned list will be filled with `float` s. Well, the word "must" is rather strong. Type hints, as of Python 3.7, have no bearing on the execution of a Python program. They truly are just "hints" rather than "musts." At runtime, a Python program can completely ignore its type hints and break any of its supposed constraints. However, type checker tooling can evaluate the type hints in a program during development, and tell the programmer if there are any illegitimate calls to a function. So a call of `repeat("hello", 30)` can be caught before it enters production (since `"hello"` is not a `float`). Let's look at one more example. This time, we will examine a type hint for a variable declaration.

```
myStrs: List[str] = repeat(4.2, 2)
```

copy
That type hint does not make sense. It says that `myStrs` is expected to be a list of strings. However, we know from the previous type hint that `repeat()` returns a list of floats. Again, because Python, as of version 3.7, does not verify type hints for correctness during execution, this mistaken type hint will have no effect on the running of the program. However, also once again, a type checker could catch this programmer error/misconception about the right type before it became a disaster.

C.2 Why are type hints useful?

Now that you know what type hints are, you may be wondering why all of this trouble is worth it. After all, you have also learned that type hints are ignored by Python at runtime. Why would one spend all this time adding type annotations to the code when the Python interpreter couldn't care less? As has already been touched upon, type hints are a good idea for two primary reasons: they self-document the code and they allow a type checker to verify the correctness of a program prior to execution.

In most programming languages with static typing (like Java or Haskell), required type declarations make it very clear what parameters a function (or method) expects and what type it will return. This alleviates some of the documentation burden on the programmer. For instance, it is completely unnecessary to specify what the following Java method expects as parameters or return type.

```
1
2


public float eatWorld(World w, Software s) { … }
```

copy
Contrast this with the required documentation for the equivalent method written in traditional Python, without type hints.

```
1
2
3
4
5
6
7
```

```
def eat_world(w, s):
```

copy
By allowing us to self-document our code, type hints make Python documentation as succinct as statically typed languages.

```
1
2


def eat_world(w: World, s: Software) -> float:
```

copy

Take it to an extreme. Imagine you inherit a code base that has no comments whatsoever. Will a comment-less codebase be easier to make sense of with type hints or without type hints? The type hints will save you from having to dig into the actual code of a comment-less function to understand what types to pass it as parameters and what type to expect it to return.

Remember, a type hint is essentially a way of stating what type is expected at some point in a program. Yet, Python does nothing to verify this expectation. That's where a type checker comes in. A type checker can take a file of Python source code filled with type hints and verify if they actually will hold when the program is run.

There are multiple different type checkers for Python type hints. For example, the popular Python IDE PyCharm has a type checker built-in. If you edit a program in PyCharm with type hints, it will automatically point out type errors. This will help you catch your mistakes before you even finish writing a function.

The leading Python type checker, as of writing the of this book, is mypy. The mypy project is led by Guido van Rossum, the same person who originally created Python itself. Does that leave any doubt in your mind that type hints could potentially have a very prominent role in the future of Python? After you install mypy, using it is as simple as running `mypy example.py` where `example.py` is the name of the file you want to type check. mypy will spit out to the console all of the type errors in your program, or nothing if there are no errors.

There may be other reasons type hints are useful in the future. Right now, type hints have no impact on the performance of a running Python program (to reiterate one last time, they are ignored at runtime). However, it is possible that future versions of Python will use the type information from type hints to perform optimizations. In such a world, perhaps you will be able to speed up the execution of your Python program by simply adding type hints. This is pure speculation of course. I know of no plans to implement type hint based optimizations in Python.

C.3 What are the downsides of type hints?

There are three potential downsides of using type hints: Code with type hints takes longer to write than code without type hints; Type hints can arguably lead to a decrease of readability in some cases; Type hints are still not fully baked and implementing some typing constraints with Python's current implementations can be confusing.

Code with type hints takes longer to write for two reasons: it's simply more typing (literally more keys hit on the keyboard) and you have to reason more about your code. Reasoning about your code is almost always a good thing. Doing extra reasoning will still slow you down. However, you will hopefully make up for that lost time by catching errors with a type checker before your program even runs. The time spent debugging errors that could be caught by a type checker is probably greater than the time spent reasoning about types at composition time for any complex codebase.

Some people find Python code with type hints less readable than Python code without them. The two reasons for this are probably unfamiliarity and verbosity. Any syntax you are not familiar with, is going to be less readable than a syntax you are familiar with. Type hints really do change the look of Python programs, making them potentially look unfamiliar at first. This can only be alleviated by writing and reading more Python code with type hints. The second issue, verbosity, is more fundamental. Python is famous for its concise syntax. Often the same program in Python is significantly shorter than its equivalent in another language. Python code with type hints is not as compact. It cannot necessarily as quickly be scanned by the naked eye—there's just a lot more there. The tradeoff is that the understanding of the code can be greater following the first read, even if that read takes longer. With type hints, you immediately see all of the expected types, a leg up over traditionally having to scan the code itself to understand the types, or having to read the documentation.

Finally, type hints are still in flux. While they have definitely improved since they were first introduced in Python 3.5, there are still edge cases in which type hints do not work well. An example of this is in chapter 2. The `Protocol` type, usually an important part of a type system, is not yet in the Python standard library's typing module, so it was necessary in chapter 2 to include the third party `typing_extensions` module. There are plans to include `Protocol` in a future version of the official Python standard library, but the fact that it's not included is a testament to the fact that these are still early days for type hints in Python. Throughout the writing of this book, I ran in to several such edge cases that were confusing to solve given the existing primitives available in the standard library. Since type hints are not required in Python, it is okay at this stage to just ignore them in areas where they would be inconvenient to use. You can still get some benefit by using type hints "half-way."

C.4 Getting more information

Every chapter of this book is filled with examples of type hints. However, it is not a tutorial on using type hints. The best place to get started with type hints is Python's official documentation for the typing module (https://docs.python.org/3/library/typing.html). That documentation explains not only all of the different built-in types that are available, but also how to use them for several advanced scenarios, beyond the scope of this brief introduction.

The other type hint resource that you should really check out is the mypy project (http://mypy-lang.org). mypy is the leading Python type checker. In other words, it is the piece of software that you will use to actually verify the validity of your type hints. Beyond installing it and using it, you should also checkout mypy's documentation (https://mypy.readthedocs.io/). The documentation is rich and explains how to use type hints in some scenarios that the standard library's documentation does not. For instance, one particularly confusing area is generics. The mypy generics documentation is a good starting point. Another nice resource is the "type hints cheat sheet" put out by mypy (https://mypy.readthedocs.io/en/stable/cheat_sheet_py3.html).