Type Hints and Overloaded Functions

One obvious place in your code that I think Type Hints would work great in are overloaded functions. We learned about function overloads back in Chapter 4. Let's grab the overloaded adding function from that chapter and take a look at how much better it will be with type hints. Here's the original code:

```
from functools import singledispatch


@singledispatch
def add(a, b):
    raise NotImplementedError('Unsupported type')


@add.register(int)
def _(a, b):
    print("First argument is of type ", type(a))
    print(a + b)


@add.register(str)
def _(a, b):
    print("First argument is of type ", type(a))
    print(a + b)


@add.register(list)
def _(a, b):
    print("First argument is of type ", type(a))
    print(a + b)
```

This example's first argument is pretty obvious if you understand how Python's function overloads work. But what we don't know is what the second argument is supposed to be. We can infer it, but in Python it is almost always best to be explicit rather than implicit. So let's add some type hints:

```
from functools import singledispatch


@singledispatch
```

```
def add(a, b):
    raise NotImplementedError('Unsupported type')


@add.register(int)
def _(a: int, b: int) -> int:
    print("First argument is of type ", type(a))
    print(a + b)
    return a + b


@add.register(str)
def _(a: str, b: str) -> str:
    print("First argument is of type ", type(a))
    print(a + b)
    return a + b


@add.register(list)
def _(a: list, b: list) -> list:
    print("First argument is of type ", type(a))
    print(a + b)
    return a + b
```

Now that we've added some type hints, we can tell just by looking at the function definition what the arguments should be. Before type hints were added, you would have needed to mention the argument types in the function's docstring, but because we have these handy hints, we don't need to clutter our docstring with that kind of information any longer.

I have worked with some poorly documented Python code before and even if it only had type hints in the function and method definitions the code would have been much easier to figure out. Type hints aren't a replacement for good documentation, but they do enhance our ability to understand our code in the future.

Wrapping Up

When I first heard about type hinting, I was intrigued. It's a neat concept and I can definitely see uses for it. The first use case that springs to my mind is just self-documenting your code. I've worked on too many code bases where it's difficult to tell what a function or class accepts and while type hinting doesn't enforce anything, it would certainly bring some clarity to some of that ambiguous code. It would be neat if some Python IDEs added an optional flag that could check your code's type hints and make sure you're calling your code correctly too.

I highly recommend checking out the official documentation as there's a lot

I highly recommend checking out the official documentation as there's a lot more information there. The PEPs also contain a lot of good details. Have fun and happy coding!