Introduction to Type Hinting with Python 3

North Austin Pythonistas
December 2019

Goals

* What is type hinting?
* What are the benefits of type hinting?
* How do I begin annotating my code?

Wait, What Is a “type” and a “hint”?

* Type == class
* Hint == Annotation
* Common python classes include:

    int, float, str, list, dict and set

* All user-defined classes too.

Type Hinting is for Variables

```python
n: int = 1
f: float = 1.0
s: str = "foo"
d: dict = {}
l: list = []
S: set = set()
t: tuple = ()
```

* A declaration ignored by the Python compiler
* Records the programmer's intent
* Appended to a variable on it's first use

VARIABLE : CLASS

VARIABLE : CLASS = VALUE

Type Hinting is Also for Functions

```
1 def function(foo: str, bar: bool = True) -> None:
2     pass
```

* Declare arguments just like variables
* Hint the function return value using ‘-> TYPE’ before the final colon.

Annotating Class Methods

```python
class Foobar:
    def __init__(self, name: str) -> None:
        self.name: str = name
        self.value: float = 0.0

    def __str__(self) -> str:
        return f"{self.name}={self.value:05.3f}"

    @property
    def status(self) -> str:
        if self.value <= 0.0:
            return "normal"
        return "oh no!"

    def acknowledge(self, ack: bool = True) -> bool:
        if ack:
            return self.value == 0.0
        return False

```

* Never annotate “self”
* __init__ returns None
* Business as Usual

A Simple Example with an Insidious Bug

```
1 def reuntokenizer(token):
2     """re-untokenizes token"""
3     return token + 1
4
5 for arg in [int(), float(), str()]:
6     result = reuntokenizer(arg)
7     print(type(result), result)
```

Executing The Insidious Bug

```
1 def reuntokenizer(token):
2     """re-untokenizes token"""
3     return token + 1
4
5 for arg in [int(), float(), str()]:
6     result = reuntokenizer(arg)
7     print(type(result), result)
```

```
1 $ python3 broke.py
2 <class 'int'> 1
```

Executing The Insidious Bug (continued)

```
1 def reuntokenizer(token):
2     """re-untokenizes token"""
3     return token + 1
4
5 for arg in [int(), float(), str()]:
6     result = reuntokenizer(arg)
7     print(type(result), result)
```

```
1 $ python3 broke.py
2 <class 'int'> 1
3 <class 'float'> 1.0
```

Executing The Insidious Bug (oh crap!)

```
1 def reuntokenizer(token):
2     """re-untokenizes token"""
3     return token + 1
4
5 for arg in [int(), float(), str()]:
6     result = reuntokenizer(arg)
7     print(type(result), result)
```

```
1 $ python3 broke.py
2 <class 'int'> 1
3 <class 'float'> 1.0
4 Traceback (most recent call last):
5   File "broke.py", line 7, in <module>
6     result = reuntokenizer(arg)
7   File "broke.py", line 3, in reuntoke
8     return token + 1
9 TypeError: can only concatenate str (
```

Welp, That's Broke

```
$ python3 broke.py
<class 'int'> 1
<class 'float'> 1.0
Traceback (most recent call last):
  File "broke.py", line 7, in <module>
    result = reuntokenizer(arg)
  File "broke.py", line 3, in reuntokenizer
    return token + 1
TypeError: can only concatenate str (not "int") to str
```

Adding Type Hinting to Reuntokenizer

```
1 def reuntokenizer(token: int) -> int:
2     """re-untokenizes token"""
3     return token + 1
4
5
6 for arg in [int(), float(), str()]:
7     result = reuntokenizer(arg)
8     print(type(result), result)
```

Still Broke, What Gives?

```
$ python3 hinted.py
<class 'int'> 1
<class 'float'> 1.0
Traceback (most recent call last):
  File "hinted.py", line 7, in <module>
    result = reuntokenizer(arg)
  File "hinted.py", line 3, in reuntokenizer
    return token + 1
TypeError: can only concatenate str (not "int") to str
```

Enter mypy, the Python 3 type checker

```
$ python3 -m pip install mypy
$ mypy hinted.py
hinted.py:6: error: Argument 1 to "reuntokenizer" has
  incompatible type "object"; expected "int"
Found 1 error in 1 file (checked 1 source file)
```

```
def reuntokenizer(token):
    """re-untokenizes token"""
    return token + 1

for arg in [int(), float(), str()]:
    result = reuntokenizer(arg)
    print(type(result), result)
```

Resources & Links

* https://docs.python.org/3.7/library/typing.html
* https://mypy.readthedocs.io/en/latest
* https://mypy.readthedocs.io/en/latest/cheat_sheet_py3.html
* https://realpython.com/python-type-checking/
* https://pre-commit.com
* https://black.readthedocs.io/en/stable/
* https://carbon.now.sh - used to format code examples
* https://github.com/North-Austin-Pythonistas/Talks

One Way to Fix the Insidious Bug

```python
from typing import Any


def reuntokenizer(token: Any) -> Any:
    """re-untokenizes token"""
    return token + type(token)(1)


for arg in [int(), float(), str()]:
    result = reuntokenizer(arg)
    print(type(result), result)
```

```
$ python3 fixed.py
<class 'int'> 1
<class 'float'> 1.0
<class 'str'> 1
```

What it Looks Like When Things Are Wrong

```python
n: int = "nope"
f: float = 1
s: str = float()
d: dict = None
l: list = {}
S: set = tuple()
t: tuple = []
```

```
$ mypy broke-variables.py
broke-variables.py:1: error: Incompatible types in assignment
(expression has type "str", variable has type "int")
broke-variables.py:3: error: Incompatible types in assignment
(expression has type "float", variable has type "str")
broke-variables.py:4: error: Incompatible types in assignment
(expression has type "None", variable has type "Dict[Any, Any]")
broke-variables.py:5: error: Incompatible types in assignment
(expression has type "Dict[<nothing>, <nothing>]", variable has type
"List[Any]")
broke-variables.py:6: error: Incompatible types in assignment
(expression has type "Tuple[<nothing>, ...]", variable has type
"Set[Any]")
broke-variables.py:7: error: Incompatible types in assignment
(expression has type "List[<nothing>]", variable has type "Tuple[Any,
...]")
Found 6 errors in 1 file (checked 1 source file)
```