How to start using types in Python with Mypy

PyCon China 2020

stackbuilders

Quilotoa Lake, Ecuador 2020.

Carlos Villavicencio

Ecuadorian 🇪🇨

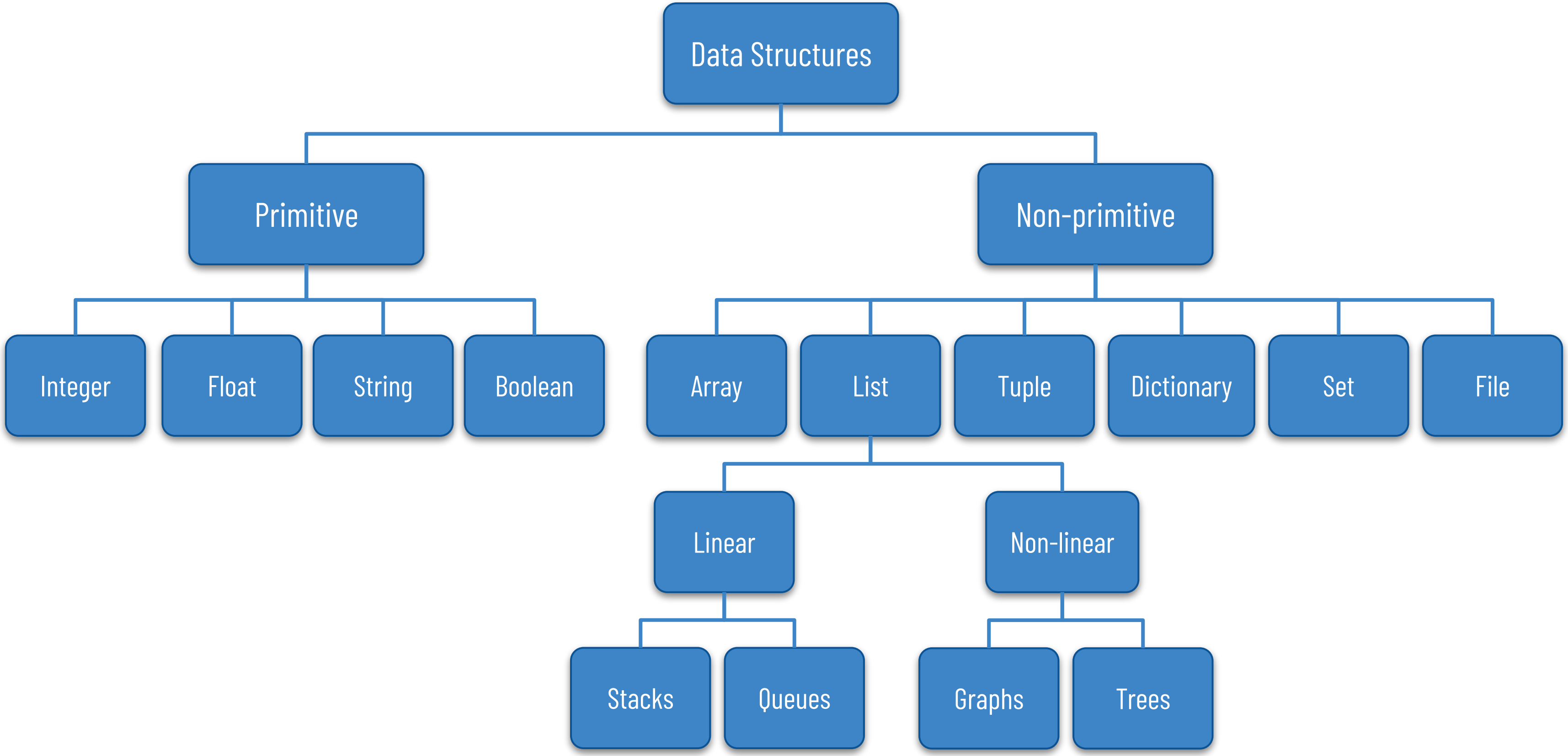
Software Developer at Stack Builders

Community leader

Amateur photographer

po5i @po5i cvillavicencio cvillavicencio@stackbuilders.com

Types and Python


Data Structures
Primitive
Non-primitive
Integer
Float
String
Boolean
Array
List
Tuple
Dictionary
Set
File
Linear
Non-linear
Stacks
Queues
Graphs
Trees

Fixing bugs on the production server

IMAGINE A WORLD
WHERE DYNAMIC LANGUAGES HAD STATIC TYPE CHECKING
imgflip.com

Dynamic vs. Static Languages

Types are known in runtime	Types are known in compilation time
Variables bind to objects 🦆	Variables bind to types
Less verbose	Very verbose
Bugs in run-time are very common	Bytecode is well optimized in memory

Weakly vs. Strongly Type System

Implicit coercion between non-related types	Explicit type conversion (**casting**)
Flexibles	Strict rules on the static analysis.
Unpredictables	Type Safety

Static type checking in Python

: my[py]

But you can also use PYRIGHT

Type Annotations

Primitive types ~~🦕~~

```
meat: str = "Beef"

weight_pounds: float = "0.5"

# mypy

# error: Incompatible types in assignment

# (expression has type "str", variable has type "float")
```

Let’s prepare some ~~🍔~~

```python
def make_hamburger(meat, number_of_meats):
  return ["bread"] + [meat] * number_of_meats + ["bread"]


print(make_hamburger("BEEF", 2))
# ['bread', 'BEEF', 'BEEF', 'bread']
```

Unit testing?

```
class MyTest(unittest.TestCase):
    def test_make_hamburger_returns_list(self):
        self.assertTrue(isinstance(make_hamburger("beef", 2), list))

    def test_empty_make_hamburger_returns_breads(self):
        self.assertEqual(make_hamburger(None, 0), ['bread', 'bread'])

    def test_invalid_make_hamburger_raises(self):
        with self.assertRaises(TypeError):
            make_hamburger()
```

The `typing` module

```
from typing import List


def make_hamburger(meat: str, number_of_meats: int) -> List[str]:
  return ["bread"] + [meat] * number_of_meats + ["bread"]
```

Type Alias

```
from typing import List


Hamburger = List[str]


def make_hamburger(meat: str, number_of_meats: int) -> Hamburger:
  return ["bread"] + [meat] * number_of_meats + ["bread"]
```

Optionals ~~🍅~~

```python
from typing import List, Optional

Hamburger = List[str]
Extras = Optional[List[str]]

def make_hamburger(meat: str, number_of_meats: int, extras: Extras) -> Hamburger:
  if extras:
    return ["bread"] + extras + [meat] * number_of_meats + ["bread"]
  else:
    return ["bread"] + [meat] * number_of_meats + ["bread"]

print(make_hamburger("Beef", 2, ['tomatoes', 'pickles']))
# ['bread', 'tomatoes', 'pickles', 'Beef', 'Beef', 'bread']
```

Generics

```
from typing import TypeVar, List

T = TypeVar("T", int, List[str])

def generic_add(x: T, y: T) -> T:
  return x + y
```

Generics 👩🏻‍💻

```
x1: int = 5
y1: int = 2
print(generic_add(x1, y1))  # 7

x2: List[str] = ["Hello"]
y2: List[str] = ["World"]
print(generic_add(x2, y2))  # ['Hello', 'World']

x3: str = "foo"
y3: str = "bar"
print(generic_add(x3, y3))  # mypy error: Value of type variable "T" of
"generic_add" cannot be "str"
```

Union Types

```
from typing import Union


Number = Union[float, int]


def union_add(x: Number, y: Number) -> Number:
    return x + y
```

Union Types

```
x1: int = 5
y1: float = 2.5
print(union_add(x1, y1))
# 7
x2: int = 2
y2: str = "1"
print(union_add(x2, y2))
# error: Argument 1 to "union_add" has incompatible type "str";
expected "Union[float, int]"
# error: Argument 2 to "union_add" has incompatible type "str";
expected "Union[float, int]"
```

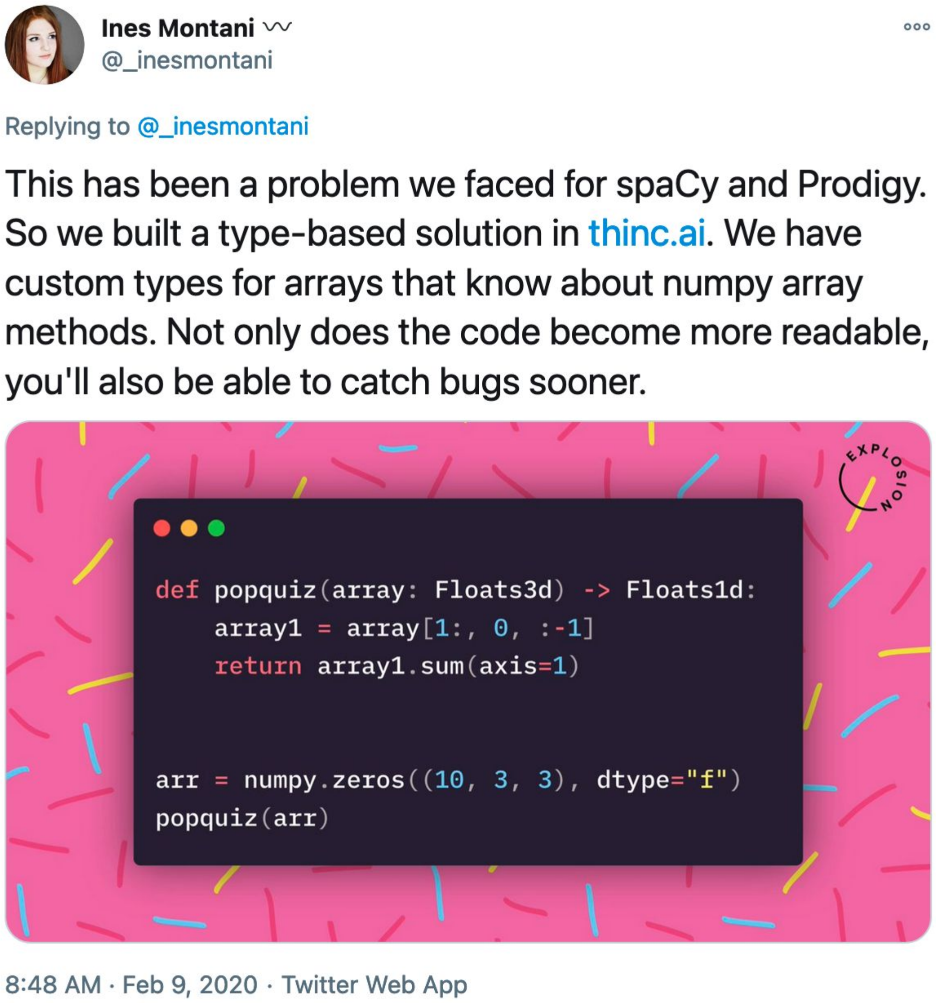
Callables

```python
from typing import Callable

def sum_and_process(a: int, b: int, callback: Callable[[int], bool]) -> bool:
  total = a + b
  return callback(total)

def is_positive(val: int) -> bool:
  return val > 0

output = sum_and_process(5, 2, is_positive)
print(output)
# True
```



Ines Montani
@_inesmontani

Replying to @_inesmontani

This has been a problem we faced for spaCy and Prodigy. So we built a type-based solution in thinc.ai. We have custom types for arrays that know about numpy array methods. Not only does the code become more readable, you'll also be able to catch bugs sooner.

```
def popquiz(array: Floats3d) -> Floats1d:
    array1 = array[1:, 0, :-1]
    return array1.sum(axis=1)


arr = numpy.zeros((10, 3, 3), dtype="f")
popquiz(arr)
```

8:48 AM · Feb 9, 2020 · Twitter Web App

stackbuilders

Check out my tutorial

Obrigado!

¡Gracias!

谢谢!

Thank You!

ευχαριστώ