Python Type System

PEP 483 - The Theory of Type Hints

PEP 484 - Type Hints

1. Type vs Class

파이썬의 타입 시스템을 이해하려면 Type 과 Class 을 구별해야한다.

type : type checker concept
class: runtime concept

type 은 runtime of "realm" 에 있는 것이 아니다.
type 은 another "layer" of a program.
type checker 를 위한 layer

type checker - a tool that analyses code
코드를 실행하지 않고 코드에서 type consistency 를 static 하게 검사

ex) mypy, pyre-check, pytype

1.1 How to define a type?

1. by defining a class
2. by specifying functions that work with variables a type
3. by using more basic types to create more complex ones

1. by defining a class

`class Animal: ...` defines `Animal` class and `Animial` type at the same time.

In this case inheritance relationships between classes are mapped one-to-one to subtyping relationships.
`Dog` is a subclass of `Animal`
`Dog` is a subtype of `Animal`

This approach to typing is called **nominal subtyping**

2. by specifying functions that work with variables a type

In the spirit of duck typing

ex) if an object has `__len__` method then it has `Sized` type.

This approach to typing is called **structural subtyping**

3. by using more basic types to create more complex ones

Use earlier defined types to define more complex types

2. Type annotation syntax

2.1 Annotating variables

```
name: Type = initial_value
```

```
width: int
width = 15  # no mypy error

height: int
height = '25'  # error:
# Incompatible types in assignment (expression has type "

depth: int = 15.5  # error:
# Incompatible types in assignment (expression has type "
```

2.2 Annotating functions

```
def function(arg1: Type1, arg2: Type2) -> ReturnType:
```

```
def add_ints(x: int, y: int) -> int:
    return x + y  # no mypy error

add_ints(1, 2)  # no mypy error
add_ints(1, 2.0)  # error:
# Argument 2 to "add_ints" has incompatible type "float";

def broken_add(x: int, y: int) -> str:
    return x + y  # error:
    # Incompatible return value type (got "int", expected
```

3. Subtyping

Understand the basic subtyping relationship.

```python
class Animal:
        ...

class Dog(Animal):
        ...
```

subtype is a less general type.
`Dog` is less general than `Animal`

Let’s dive a bit deeper and see how subtyping relation is defined in Python.

3.1 Definition

`<:` mean "is a subtype of".
`B <: A` reads "`B` is a subtype of `A`".

1. every value of type `B` is also in the set of values of type `A`
2. every function of type `A` is alos in the set of functions of type`B`

`Dog <: Animal`

1. Set of `Dog`s is a subset of `Animal`s (every `Dog` is an `Animal`, but not every `Animal` is a `Dog`
2. Set of functions of `Animal` is a subset of functions of `Dog` (`Dog` can do whatever `Animal` can, but `Animal can't do everything`Dog` can)

3.2 Assignment rules

```python
# Dob <: Animal
scooby: Dog
an_animal: Animal

an_animal = scooby  # no mypy error
```

Assigning `scooby` to `an_animal` is *type-safe* because `scooby` is guaranteed to be an `Animal`.

```
# Dog <: Animal
scooby: Dog
an_animal: Animal

scooby = an_animal  # error:
# Incompatible types in assignment (expression has type "
```

Not type-safe because `an_animal` might not be a `Dog`.

3.3 Attribute rules

Mypy checks if an attribute is actually defined on an object.

```
class Animal:
        def eat(self): ...

class Dog(Animal):
        def bark(self): ...
```

```
# Dog <: Animal
an_animal: Animal
snoopy: Dog

an_animal.eat()  # no mypy error
snoopy.eat()  # no mypy error

snoopy.bark()  # no mypy error
an_animal.bark()  # error: "Animal" has no attribute "bar
```

4. Defining complex types

4.1 List

`List[TypeOfElements]`

```
from typing import List

my_list: List[int] = [1, 2, 3]

my_other_list: List[int] = [1, 2, '3']
# error: List item 2 has incompatible type "str"; expecte

class Animal: pass
class Dog(Animal): pass

scooby = Dog()
lassie = Dog()
pinky = Animal()

my_dogs: List[Dog] = [scooby, lassie, pinky]
# error: List item 2 has incompatible type "Animal"; expe
```

4.2 Tuple

Python language `tuple` has two purposes.

1. immutalbe list: `Tuple[TypeOfAllElements, ...]`
2. record or row of values: `Tuple[Type1, Type2, Type3]`

```python
from typing import Tuple

bob: Tuple[str, str, int] = ('Bob', 'Smith', 25)  # no my

frank: Tuple[str, str, int] = ('Frank', 'Brown', 43.4)  #
# Incompatible types in assignment (expression has type "
#   variable has type "Tuple[str, str, int]")

ann: Tuple[str, str, int] = ('Ann', 'X', 1, 2)  # error:
# Incompatible types in assignment (expression has type "
#   variable has type "Tuple[str, str, int]")

scores1: Tuple[int, ...] = (5, 8, 4, -1)  # no mypy error

scores2: Tuple[int, ...] = (5, 8, 4, -1, None, 7)  # erro
# Incompatible types in assignment (expression has type
#   "Tuple[int, int, int, int, None, int]", variable has
```

4.4 Dict

`Dict[KeyType, ValueType]`

```
from typing import Dict

id_to_name: Dict[int, str] = {1: 'Bob', 23: 'Ann', 7: 'Ka
id_to_age: Dict[int, int] = {'1': 41, 2: 22}  # error:
# Dict entry 0 has incompatible type "str": "int"; expect

name_to_phone_no: Dict[str, str] = {'Bob': '55534534', 'A
# Dict entry 1 has incompatible type "str": "int"; expect
```

4.5 Union

`Union[Type1, Type2, Type3]` docs

```
from typing import Union

width1: Union[int, float] = 20 # or 20.5
width3: Union[int, float] = '44'

class Animal:
        def eat(self): pass

class Dog(Animal): pass
class Cat(Animal): pass
class Lizard(Animal): pass

def restricted_eat(animal: Union[Dog, Cat]) -> None:
        animal.eat()

a_dog: Dog
restricted_eat(a_dog)

a_cat: Cat
restricted_eat(a_cat)
```

4.6 None Type and Optional Type

In python, `None` symbolize no value. Type of `None` is `NoneType`, but in typing context, there is an alias for it, which is... None itself.

value of type T or no value type would be `Union[T, None]`. So *int or nothing* would be `Union[int, None]`.

`Union[T, None]` has an alias: `Optional[T]`

something or nothing patern

```python
from typing import Optional

def get_user_id() -> Optional[int]:
        pass

def process_user_id(user_id: int):
        pass

user_id = get_user_id()
process_user_id(user_id)
# error: Argument 1 to "process_user_id" has incompatible
```

Reference

First Steps with Python Type System

Next Steps with Python Type System

PEP 483 - The Theory of Type Hints

PEP 484 - Type Hints

typing — Support for type hints

What are data classes and how are they different from common classes?

Type Systems: Structural vs. Nominal typing explained

mypy와 함께하는 Python Typing

Python typing으로 인한 순환 참조 대응책