

# Learn static type check the hard way

WPH95@Dashbase

# ME

前狂热 Django 使用者  
过气 OJ – CodeVS 维护者

[github.com/WPH95](https://github.com/WPH95)

一个被逼用了一段时间 JAVA 的程序猿🙈  
重新用回 python 遇到的苦恼

# 房间里的两个是什么？

```
public class Home {  
    public class Cat {  
        public void meow() {  
            }  
    }  
}  
  
public class Mouse {  
    public void squeak() {  
    }  
}  
  
public static Home of(Cat tom, Mouse jerry) {  
    tom.meow();  
    jer  
    p • jerry  
    Press ^Space to see non-imported classes >>  
}
```

Mouse

# 房间里的两个是什么？

```
class Cat(object):
    def meow(self):
        pass

class Mouse(object):
    def squeak(self):
        pass

def home(tom, jerry):
    tom.
```

main      if \_\_name\_\_ == '\_\_ma  
if  
ifn      if e  
ifnn      if expr  
not

# 房间里的两个是什么？

```
class Cat(object):  
    def meow(self):  
        pass  
  
class Mouse(object):  
    def squeak(self):  
        pass  
  
def home(tom: Cat, jerry):  
    tom.  
        meow(self)          Cat  
        __dict__            object  
        __init__(self)      object
```

# Items?

```
def add(self, items):  
    for item in items:  
        self.wait_list.append(item.ip)
```

# Items?

```
from typing import List
from youproject.modles import User

def add(self, items: List[User]):
    for item in items:
        self.wait_list.append(item.ip)
```



static type check?

# Type Hint ?

*noun* • **UK**  /hint/ **US**  /hint/

**hint** *noun* (INDIRECT  
STATEMENT)

★ **B2** [C] something that you say  
or do that shows what you think  
or want, usually in a way that is  
not direct

暗示, 提示, 示意



使用 `type hint && check`

python type-check 历史

All Things Pythonic

## Adding Optional Static Typing to Python

by Guido van van Rossum

December 23, 2004

### Summary

Optional static typing has long been requested as a Python feature. It's been studied in depth before (e.g. on the type-sig) but has proven too hard for even a PEP to appear. In this post I'm putting together my latest thoughts on some issues, without necessarily hoping to solve all problems.

---

An email exchange with Neal Norwitz that started out as an inquiry about the opening of a stock account for the PSF (talk about bizarre conversation twists) ended up jogging my thoughts about optional static typing for Python.

[As an experiment, I'm going to post this to Artima without mentioning it anywhere else. If RSS works, it should show up on various other blogs within days.]

## What is Optional Static Typing?

The Python compiler doesn't know about the types of the objects you pass around in a Python program; only the run-time (the Virtual Machine) does. This makes Python expressive and flexible, but sometimes means that bugs of a rather trivial kind (like typos in method names) are found later than the developer would have liked. Without losing the benefits of Python's dynamic typing, it would be nice if you had the option to add type declarations for your method arguments, variables and so on, and then the compiler would give you a warning if you did something that wasn't possible given what the compiler knows about those types. While there are third-party program checkers that find a lot of problems without type declarations, e.g. [pychecker](#), it would be nice if (a) this capability was built into the Python compiler and (b) you could give it hints in cases where its type inference broke down.

Let's look at a simple function:

```
def gcd(a, b):  
    while a:  
        a, b = b%a, a
```

<https://www.artima.com/weblogs/viewpost.jsp?thread=85551>

So let's consider a simple type annotation for this function:

```
def gcd(a: int, b: int) -> int:  
    while a:  
        a, b = b%a, a  
    return b
```

I've considered various ways of adding argument types, and I've found it ambiguous to distinguish between these two:

```
def foo(): int:
```

```
def foo(): print
```



```
>> def foo(a:'x', b: 5 + 6, c:list)-> max(2, 9):  
>>     pass  
  
>> foo.__annotations__  
<< {'a': 'x', 'b': 11, 'c': list, 'return': 9}
```

## PEP 3107 -- Function Annotations

Created: 2006-12-02

PEP 484 -- Type Hints [3.5]

PEP 526 -- Syntax for Variable Annotations [3.6]

PEP 544 -- Protocols: Structural subtyping (static duck typing) [3.7]

PEP 561 -- Distributing and Packaging Type Information [3.7]



**Mypy** Dropbox 2014–now

an experimental optional **static type checker** for Python that aims to combine the benefits of dynamic (or "duck") typing and static typing.

**Pyre** Facebook 2018–now

Performant type-checking for python.

**Parallel, Million lines code**

## PEP 3107 -- Function Annotations

1. Function annotations, both for parameters and return values, are **completely optional**.

2. Function annotations are nothing more than a way of associating arbitrary Python expressions with various parts of a function at **compile-time**

3. This work will be left to third-party libraries.

**type hint**

PEP3107 [3.0]  
PEP484 [3.5]  
PEP526 [3.6]  
PEP561 [3.7]

---

**static type check**

Mypy [Dropbox]  
Pyre [Facebook]

# 如何在代码中注释类型

## 简单的例子🍎

```
def say_hello(name: str) -> str:  
    return 'Hello ' + name
```

Python version >= 3.5

## 简单的例子🍎

```
def say_hello(name: str) -> str:  
    return 'Hello ' + name
```

```
def say_hi(name):  
    """  
    :type name: str  
    :rtype str  
    """  
    return "hi"
```

# Simple Type check

```
def say_hello(name: str) -> str:  
    return 'Hello ' + name
```

```
say_hello("wph95")  
say_hello(b"wph95")  
say_hello(95)
```

```
>> mypy hello.py
```

```
:14: error: Argument 1 to "say_hello" has incompatible type "bytes"; expected "str"  
:15: error: Argument 1 to "say_hello" has incompatible type "int"; expected "str"
```



# Class

```
from typing import List

class Room:
    def __init__(self, users: List[str]) -> None:
        self.users = []
        for user in users:
            self.users.append(user)

    def is_here(self, name: str) -> bool:
        return name in self.users
```

# Callable

[ [Arg1Type, Arg2Type], ReturnType]

```
from typing import Callable
```

```
def feeder(get_next_item: Callable[[], str]) -> None:  
    pass
```

```
def async_query(on_success: Callable[[int], None],  
                on_error: Callable[[int, Exception], None]) -> None:  
    pass
```

```
def partial(func: Callable[..., str], *args) -> Callable[..., str]:  
    pass
```

# Union

```
from typing import Union

def flip() -> Union[Head, Tail]:
```

# Union

```
from typing import Union  
  
def is_here() -> Union[str, None]:
```

# Optional

```
from typing import Optional  
  
def is_here() -> Optional[str]:
```

# @overload

```
from typing import Optional, overload
```

```
@overload
```

```
def pong(text: None) -> None:  
    pass
```

```
@overload
```

```
def pong(text: str) -> str:  
    pass
```

```
def pong(text: Optional[str]) -> Optional[str]:  
    if not text:  
        return None  
    return "pong"
```

**Generic**

# Generic

```
from typing import TypeVar, Generic
```

```
Numeric = TypeVar("Numeric", int, float)
```

```
def add(a: Generic[Numeric], b: [Numeric]) -> Generic[Numeric]:  
    print()  
    print(type(a), type(b))  
    return a + b
```



# Generic

```
from typing import TypeVar, Generic, List
```

```
T = TypeVar('T')
```

```
class Stack(Generic[T]):
```

```
    def __init__(self) -> None:  
        self.items: List[T] = []
```

```
    def push(self, item: T) -> None:  
        self.items.append(item)
```

```
    def pop(self) -> T:  
        return self.items.pop()
```

# Duck Type

```
class Duck:
    def fly(self):
        print("Duck flying")

class Airplane:
    def fly(self):
        print("Airplane flying")

class Whale:
    def swim(self):
        print("Whale swimming")

def lift_off(entity):
    entity.fly()
```

# Protocol

```
from typing import Protocol

class CanFly(Protocol):
    @abstractmethod
    def fly(self) -> None:
        raise NotImplementedError

def lift_off(entity: CanFly):
    entity.fly()
```

# Stub Files

.pyi            给你的 python 文件打上一个 type hint 补丁 :)

The screenshot shows the GitHub interface for the `python/typeshed` repository. The file path `typeshed / third_party / 2and3 / click / core.pyi` is highlighted. A commit by `euresti` is shown with the message "Click: Make group and command decorators return the correct types (#2331)". Below this, the file size is listed as 11.1 KB. The code snippet shows imports from `contextlib` and `typing`, with type hints for `Any`, `Callable`, `Dict`, `Generator`, and `Iterable`.

python / **typeshed** Watch 46 Star 787 Fork 460

Code Issues 101 Pull requests 12 Projects 0 Insights

Branch: master ▾ **typeshed** / **third\_party** / **2and3** / **click** / **core.pyi** Find file Copy path

**euresti** Click: Make group and command decorators return the correct types (#2331) 99f25d0 15 days ago

7 contributors

471 lines (371 sloc) | 11.1 KB Raw Blame History

```
1 from contextlib import contextmanager
2 from typing import (
3     Any,
4     Callable,
5     Dict,
6     Generator,
7     Iterable,
```

# Future

**MonkeyType** generates static type annotations  
by collecting runtime types

**pyannotate** Auto-generate PEP-484 annotations