



You have 1 free member-only story left this month. [Sign up for Medium and get an extra one](#)



Unsplash

Dynamic typing, which means that the type of a variable is determined only during runtime, makes Python flexible and convenient.

However, every coin has two sides. Messy typing of variables is usually the root of hidden bugs. Robust programs should be type-safe.

Therefore, Python provides the typing hint feature since version 3.5.

This feature is a good trade-off. It merely provides hints instead of changing Python into a static typing language. But it can make your Python programs:

- More readable for other developers
- Type safer to avoid hard-to-find bugs
- Provide more information for modern IDEs' automatic code completion
- IDEs' can help you check the potential bugs with the help of the type hints

Let's see why it can make your programming life easier.

For example, the following function has an argument called `data`. You know it should be a string. But when you need to use some built-in methods for a string, the IDE (PyCharm in this case) can't give you any clue.

```
def clean_data(data):  
    data.  
    ifn if expr is None  
    not not expr  
    par (expr)  
    if if expr  
    ifnn if expr is not None  
    main if __name__ == '__main__': expr  
    print print(expr)  
    return return expr  
    while while expr  
    Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards Next Tip
```

It's reasonable. Because the IDE doesn't know what the type of the `data` variable is `string`. We need to add a hint here:

```
def clean_data(data:str):  
    data.  
    title(self) str  
    find(self, __sub, __start, __end) str  
    join(self, __iterable) str  
    count(self, x, __start, __end) str  
    index(self, __sub, __start, __end) str  
    capitalize(self) str  
    casefold(self) str  
    center(self, __width, __fillchar) str  
    encode(self, encoding, errors) str  
    endswith(self, __suffix, __start, __end) str  
    expandtabs(self, tabsize) str  
    format(self, __args, kwargs) str  
    Ctrl+Down and Ctrl+Up will move caret down and up in the editor. Next Tip
```

As the above screenshot shows, after adding the typing hint, the IDE knows the `data` variable is a string and displays all built-in string methods for our reference.

This makes our programming experience happier. If we forget a special string method of Python, we mostly don't need to google it since PyCharm will display all choices for us.

Now, let's dive into more tricks about type hints in Python.

## 1. Type Hints for Basic Data Types

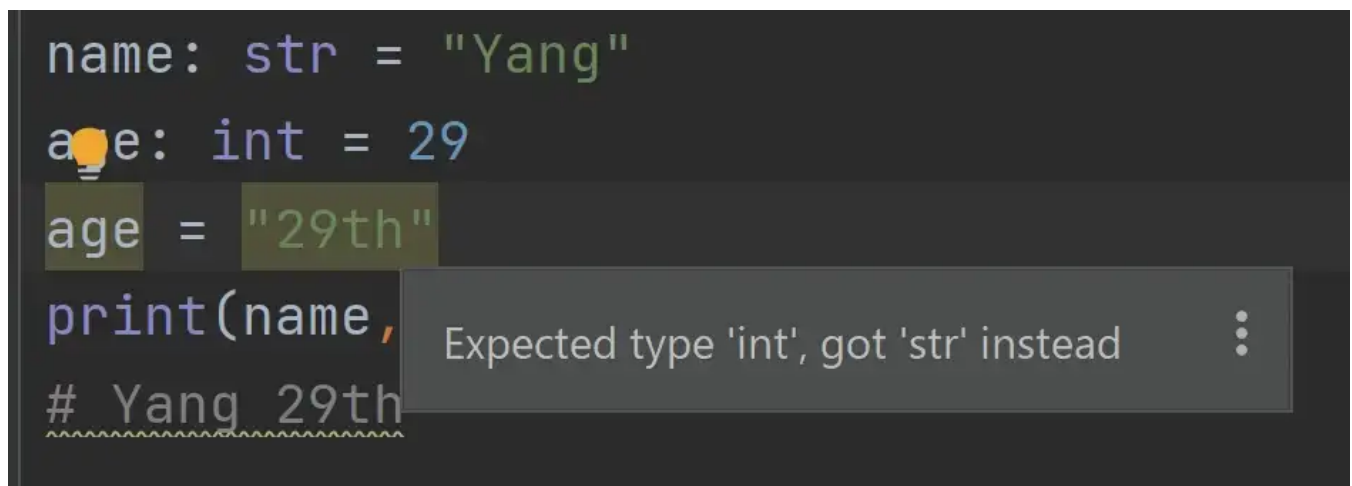
Fundamentally, when we define a variable, we can add the `:type` suffix after its name to give it a type hint.

```
name: str = "Yang"  
age: int = 29  
assets: list = []  
posts: tuple = ()  
hobbies: dict = {}
```

As mentioned before, these are just hints and not mandatory rules. If you break it, everything will be still okay:

```
name: str = "Yang"
age: int = 29
age = "29th"
print(name, age)
# Yang 29th
```

However, it's a bad practice and the IDE will remind you:



```
name: str = "Yang"
age: int = 29
age = "29th"
print(name, age)
# Yang 29th
```

Expected type 'int', got 'str' instead

*Note: Some older Python version doesn't support to use the `list` type hint directly. You need to import it from the `typing` module:*

```
from typing import List
```

Besides, we can define the types inside an iterator. Such as a list containing strings:

```
posts: list[str] = ['Python Type Hints', 'Python Tricks']
```

We can also provide type hints for a dictionary's keys and values:

```
posts: dict[int, str] = {1: 'Python Type Hints', 2: 'Python Tricks'}
```

## 2. Define a Constant Using Final Type

Python doesn't have an internal mechanism for constants. In practice, we usually name a constant in all caps to separate it from other variables.

Since Python supports type hints now, we can make this clearer through the `Final` type:

```
from typing import Final

DATABASE: Final = "MySQL"
```

## 3. Add Multiple Type Hints to One Variable

Python is a dynamic typing language. We can give one variable more than one type hint for sure:

```
from typing import Union

data: Union[int, float] = 3.14
```

The above code defined the variable `data` which can be `int` or `float` with the help of the `Union` keyword.

Besides the above approach, since Python 3.10, there is a neater way:

```
data: int|float = 3.14
```

## 4. Use General Type Hints

Python type hints also provide more general types.



For example, if we just need a variable as an iterable, which could be lists, tuples or others, we can write it as follows:

```
from typing import Iterable

def my_func(nums: Iterable):
    for n in nums:
        print(n)
```

If a variable could be `None`, we can add `None` as a possible type for it:

```
from typing import Union
a: Union[int, None] = 123
```

The above code seems ugly, so Python provides an `Optional` type hint:

```
from typing import Optional

a: Optional[int] = 123
```

As the above example shows, `Optional[X]` is equivalent to `Union[X, None]`.

Furthermore, if a variable could be any type, we can just mark it as an `Any` type:

```
from typing import Any

def my_func(nums: Any):
    pass
```

## 5. Type Hints for Functions

Using type hints in Python functions can make them more readable and self-documenting.

Let's make a simple function with type hints:

```
def greeting(name: str) -> str:
    return "Hello " + name
```

In this example, the `greeting` function takes in a string argument `name` and returns a string.

- The type hint for the argument `name` is `str`, which specifies that it should be a string.
- The syntax for defining the function's returning variable's type is `-> type`. In our example, the return type hint is `str`, which specifies that the function should return a string.

If a function's parameters are other functions, we can define them as `Callable` as the following example:

```
import time
from typing import Callable

def calc_time(func: Callable):
    start = time.time()
    func()
    end = time.time()
    print(f"Execution time: {end - start}s")

calc_time(lambda: sorted("Yang Zhou is writing!"))
```

## 6. Alias of Type Hints

For complex type hints, you probably don't like to write them again and again. In this case, Python is able to define an alias for a type hint.

```
from typing import TypeAlias

PostsType: TypeAlias = dict[int, str]

new_posts: PostsType = {1: 'Python Type Hints', 2: 'Python Tricks'}
old_posts: PostsType = {}
```

As the above example shows, we made an alias for the type `dict[int, str]` with the help of `TypeAlias`. So we just need to input the alias whenever we need this type.

The usage of `TypeAlias` is introduced from Python 3.10 and it's optional, the following code also works perfectly without it:

```
PostsType = dict[int, str]

new_posts: PostsType = {1: 'Python Type Hints', 2: 'Python Tricks'}
old_posts: PostsType = {}
```

## 7. Type Hints for a Class Itself

There is a more complicated scenario.

As the following screenshot shows, if we need type hints for a class itself when defining the class, there is an unresolved reference problem.

There are alternative ways to do the type hints for the above scenario. However, my favourite is the incoming feature of Python 3.11 (declared in [PEP 673](#)).

Since Python 3.11, we can write it as follows:

```
from typing import Self
```



```
class ListNode:
    def __init__(self, prev_node: Self) -> None:
        pass
```

## 8. Provide Literals for a Variable

Sometimes, a variable should be only assigned to some fixed values. We can use `Literal` from the `typing` module to implement this.

A **literal** is a notation in Python for representing fixed (const) values for a variable.

As the above example shows, we use `Literal` hints to let the IDE know that the `weekend` variable can't be assigned to other values except "Saturday" and "Sunday".

## Conclusion

Type hints in Python are not mandatory, but having it can make our code more readable and robust. Especially it's can give IDE more hints to help you find hidden bugs earlier.

*Thanks for reading.* ❤️

*Join Medium through my referral link to access millions of great articles:*

**Join Medium with my referral link - Yang Zhou**

Read every story from Yang Zhou (and thousands of other writers on Medium). Your membership fee directly supports Yang...

[yangzhou1993.medium.com](https://yangzhou1993.medium.com)

---

## Enjoy the read? Reward the writer.<sup>Beta</sup>

Your tip will go to Yang Zhou through a third-party platform of their choice, letting them know you appreciate their story.



Give a tip

---

## Get an email whenever Yang Zhou publishes.

Your email

---



Subscribe

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.



[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app



Download on the  
App Store



GET IT ON  
Google Play