Python Type Hints

Summary: in this tutorial, you'll learn about the python type hints and how to use the mypy tool to check types statically.

Introduction to Python type hints

Some programming languages have static typing, such as C/C++. It means that you need to declare types of variables, parameters, and return values of a function upfront. The predefined types allow the compilers to check the code before compiling and running the program.

Python uses dynamic typing, in which variables (https://www.pythontutorial.net/python-basics/python-variables/) , parameters, and return values of a function can be any type. Also, the types of variables can change while the program runs.

Generally, dynamic typing makes it easy to program and causes unexpected errors that you can only discover until the program runs.

Python's type hints provide you with optional static typing to leverage the best of both static and dynamic typing.

The following example defines a simple function (https://www.pythontutorial.net/python-basics/python-functions/) that accepts a string and returns another string:

```python
def say_hi(name):
    return f'Hi {name}'


greeting = say_hi('John')
print(greeting)
```

Here’s the syntax for adding type hints to a parameter and return value of a function:

```
parameter: type
-> type
```

For example, the following shows how to use type hints for the `name` parameter and return value of the `say_hi()` function:

```python
def say_hi(name: str) -> str:
    return f'Hi {name}'


greeting = say_hi('John')
print(greeting)
```

Output:

```
Hi John
```

In this new syntax, the `name` parameter has the `str` type:

```
name: str
```

And the return value of the `say_hi()` function also has the type `str` :

```
-> str
```

Besides the `str` type, you can use other built-in types such as `int`, `float`, `bool`, and `bytes` for type hintings.

It’s important to note that the Python interpreter ignores type hints completely. If you pass a number to the `say_hi()` function, the program will run without any warning or error:

```
def say_hi(name: str) -> str:
    return f'Hi {name}'


greeting = say_hi(123)
print(greeting)
```

Output:

```
Hi 123
```

To check the syntax for type hints, you need to use a static type checker tool.

Using a static type checker tool: mypy

Python doesn’t have an official static type checker tool. At the moment, the most popular third-party tool is Mypy. Since Mypy is a third-party package, you need to install it using the following `pip` command:

```
pip instal mypy
```

Once installed `mypy`, you can use it to check the type before running the program by using the following command:

```
mypy app.py
```

It'll show the following message:

```
app.py:5: error: Argument 1 to "say_hi" has incompatible type "int"; expected "st
Found 1 error in 1 file (checked 1 source file)
```

The error indicates that the argument of the `say_hi` is `int` while the expected type is `str` .

If you change back the argument to a string and run the `mypy` again, it'll show a success message:

```
Success: no issues found in 1 source file
```

Type hinting & type inference

When defining a variable, you can add a type hint like this:

```
name: str = 'John'
```

The type of the `name` variable is `str` . If you assign a value that is not a string to the `name` variable, the static type checker will issue an error. For example:

```
name: str = 'Hello'
name = 100
```

Error:

```
app.py:2: error: Incompatible types in assignment (expression has type "int", var
Found 1 error in 1 file (checked 1 source file)
```

Adding a type to a variable is unnecessary because static type checkers typically can infer the type based on the value assigned to the variable.

In this example, the value of the name is a literal string so that the static type checker will infer the type of the name variable as str. For example:

```
name = 'Hello'
name = 100
```

It'll issue the same error:

```
app.py:2: error: Incompatible types in assignment (expression has type "int", var
Found 1 error in 1 file (checked 1 source file)
```

Adding type hints for multiple types

The following `add()` function returns the sum of two numbers:

```
def add(x, y):
    return x + y
```

The numbers can be integers or floats. To set type hints for multiple types, you can use `Union` from the `typing` module.

First, import `Union` from `typing` module:

```
from typing import Union
```

Second, use the `Union` to create a union type that includes `int` and `float` :

```
def add(x: Union[int, float], y: Union[int, float]) -> Union[int, float]:
    return x + y
```

Here is the complete source code:

```
from typing import Union


def add(x: Union[int, float], y: Union[int, float]) -> Union[int, float]:
    return x + y
```

Starting from Python 3.10, you can use the X | Y syntax to create a union type, for example:

```
def add(x: int | float, y: int | float) -> int | float:
    return x + y
```

Type aliases

Python allows you to assign an alias to a type and use the alias for type hintings. For example:

```
from typing import Union

number = Union[int, float]


def add(x: number, y: number) -> number:
    return x + y
```

In this example, we assign the `Union[int, float]` type an alias `Number` and use the `Number` alias in the add() function.

Adding type hints for lists, dictionaries, and sets

You can use the following built-in types to set the type hints for a list (https://www.pythontutorial.net/python-basics/python-list/) , dictionary (https://www.pythontutorial.net/python-basics/python-dictionary/) , and set (https://www.pythontutorial.net/python-basics/python-set/) :

- list

- dict
- set

If you type hints a variable as a list but later assign a dictionary to it, you'll get an error:

```
ratings: list = [1, 2, 3]
ratings = {1: 'Bad', 2: 'average', 3: 'Good'}
```

Error:

```
app.py:3: error: Incompatible types in assignment (expression has type "Dict[int,
Found 1 error in 1 file (checked 1 source file)
```

To specify the types of values in the list, dictionary, and sets, you can use type aliases from the typing module:

Type Alias	Built-in Type
List	list
Tuple	tuple
Dict	dict
Set	set
Frozenset	frozenset
Sequence	For list, tuple, and any other sequence data type.
Mapping	For dictionary (dict), set, frozenset, and any other mapping data type
ByteString	bytes, bytearray, and memoryview types.

For example, the following defines a list of integers:

```python
from typing import List

ratings: List[int] = [1, 2, 3]
```

None type

If a function doesn’t explicitly returns a value, you can use None to type hint the return value. For example:

```python
def log(message: str) -> None:
    print(message)
```

Summary

- Use type hints and static type checker tools to make your code more robust.