Chapter 1

函数中的类型提示

1.1 关于渐进式类型

A gradual type system:

Is optional By default, the type checker should not emit warnings for code that has no type hints. Instead, the type checker assumes the Any type when it cannot determine the type of an object. The Any type is considered compatible with all other types.

Does not catch type errors at runtime Type hints are used by static type checkers, linters, and IDEs to raise warnings. They do not prevent inconsistent values from being passed to functions or assigned to variables at runtime.

Does not enhance performance Type annotations provide data that could, in theory, allow optimizations in the generated bytecode, but such optimizations are not implemented in any Python runtime that I am aware in of July 2021.

1.2 类型由受支持的操作定义

In a gradual type system, we have the interplay of two different views of types:

Duck typing The view adopted by Smalltalk—the pioneering object-oriented language—as well as Python, JavaScript, and Ruby. Objects have types, but variables (including parameters) are untyped. In practice, it doesn't matter what the declared type of the object is, only what operations it actually supports. If I can invoke birdie.quack(), then birdie is a duck in this context. By definition, duck typing is only enforced at runtime, when operations on objects are attempted. This is more flexible than nominal typing, at the cost of allowing more errors at runtime.

Nominal typing The view adopted by C++, Java, and C#, supported by annotated Python. Objects and variables have types. But objects only exist at runtime, and the type checker only cares about the source code where variables (including parameters) are annotated with type hints. If Duck is a subclass of Bird, you can assign a Duck instance to a parameter annotated as birdie: Bird. But in the body of the function, the type checker considers the call birdie.quack() illegal, because birdie is nominally a Bird,

and that class does not provide the .quack() method. It doesn't matter if the actual argument at runtime is a Duck, because nominal typing is enforced statically. The type checker doesn't run any part of the program, it only reads the source code. This is more rigid than duck typing, with the advantage of catching some bugs earlier in a build pipeline, or even as the code is typed in an IDE.

鸭子类型更容易上手，也更灵活，但是无法主治不受支持的操作在运行时导致错误。名义类型在运行代码之前检测错误，但有时会拒绝实际能运行的代码。（运行实例）