

For Python, many of the advantages of interface specifications can be obtained by an appropriate test discipline for components.

A good test suite for a module can both provide a regression test and serve as a module interface specification and a set of examples. Many Python modules can be run as a script to provide a simple “self test.” Even modules which use complex external interfaces can often be tested in isolation using trivial “stub” emulations of the external interface. The `doctest` and `unittest` modules or third-party test frameworks can be used to construct exhaustive test suites that exercise every line of code in a module.

An appropriate testing discipline can help build large complex applications in Python as well as having interface specifications would. In fact, it can be better because an interface specification cannot test certain properties of a program. For example, the `append()` method is expected to add new elements to the end of some internal list; an interface specification cannot test that your `append()` implementation will actually do this correctly, but it’s trivial to check this property in a test suite.

Writing test suites is very helpful, and you might want to design your code to make it easily tested. One increasingly popular technique, test-driven development, calls for writing parts of the test suite first, before you write any of the actual code. Of course Python allows you to be sloppy and not write test cases at all.

3.23 Why is there no goto?

In the 1970s people realized that unrestricted `goto` could lead to messy “spaghetti” code that was hard to understand and revise. In a high-level language, it is also unneeded as long as there are ways to branch (in Python, with `if` statements and `or`, `and`, and `if-else` expressions) and loop (with `while` and `for` statements, possibly containing `continue` and `break`).

One can also use exceptions to provide a “structured goto” that works even across function calls. Many feel that exceptions can conveniently emulate all reasonable uses of the “go” or “goto” constructs of C, Fortran, and other languages. For example:

```
class label(Exception): pass # declare a label

try:
    ...
    if condition: raise label() # goto label
    ...
except label: # where to goto
    pass
...
```

This doesn’t allow you to jump into the middle of a loop, but that’s usually considered an abuse of `goto` anyway. Use sparingly.

3.24 Why can’t raw strings (r-strings) end with a backslash?

More precisely, they can’t end with an odd number of backslashes: the unpaired backslash at the end escapes the closing quote character, leaving an unterminated string.

Raw strings were designed to ease creating input for processors (chiefly regular expression engines) that want to do their own backslash escape processing. Such processors consider an unmatched trailing backslash to be an error anyway, so raw strings disallow that. In return, they allow you to pass on the string quote character by escaping it with a backslash. These rules work well when r-strings are used for their intended purpose.

If you’re trying to build Windows pathnames, note that all Windows system calls accept forward slashes too:

```
f = open("/mydir/file.txt") # works fine!
```

If you’re trying to build a pathname for a DOS command, try e.g. one of