Python Testing (doctest & Assertions)

Table of contents

`doctest`

A **docstring test** is an `>>>` example embedded in a function's docstring. `doctest.testmod()` actually *runs* every such example in the current module and reports failures:

```
>>> import doctest
>>> doctest.testmod()
TestResults(failed=0, attempted=2)
```

`doctest.testfile(path)` instead runs every `>>>` example found in an arbitrary text file (useful for a test suite kept *outside* the functions being tested).

Doctest compares printed strings, not values

`doctest` compares Python's *exact printed output* against the expected text — not whether two values are `==`. `identity(1.0)` printing `1.0` will not match an expected `1`, even though `1.0 == 1`.

Consequences:

- Whitespace matters — `[ ]` != `[]`, and Python always prints `, ` (comma-space) inside collection literals.

- Unordered types (`set`, `dict`) don't have a guaranteed print order for non-numeric elements — compare with `==` inside the doctest instead of relying on printed order:

```
>>> {3, 1, 2} == some_function({1, 2, 3})
True
```

- Floats are inexact — compare with a tolerance instead of exact equality:

```
>>> abs(f(x) - expected) < 10**-3
True
```

Writing a comprehensive doctest

1. Typical cases *and* edge cases.
2. The zero of the data type (`0`, `[]`, `""`).
3. The singleton of the data type (`1`, `[1]`, `"a"`).
4. Correctness, not contract violations (don't test precondition failures).
5. No redundant tests.

Assertions

`assert <condition>` raises `AssertionError` (halting the program) if `<condition>` is false — useful for catching impossible situations as soon as they occur, rather than letting them silently propagate:

```
assert ans > 0   # all factorials are positive/non-zero
```

`assert False` documents a line that should be unreachable (e.g. after an exhaustive `if`/`else`).