# 18.1. Introduction: Test Cases

A **test case** expresses requirements for a program, in a way that can be checked automatically. Specifically, a test asserts something about the state of the program at a particular point in its execution.

We have previously suggested that it's a good idea to first write down comments about what your code is supposed to do, before actually writing the code. It is an even better idea to write down some test cases before writing a program.

There are several reasons why it's a good habit to write test cases.

- Before we write code, we have in mind what it *should* do, but those thoughts may be a little vague. Writing down test cases forces us to be more concrete about what should happen.
- As we write the code, the test cases can provide automated feedback. You've actually been the beneficiary of such automated feedback via test cases throughout this book in some of the activecode windows and almost all of the exercises. We wrote the code for those test cases but kept it hidden, so as not to confuse you and also to avoid giving away the answers. You can get some of the same benefit from writing your own test cases.
- In larger software projects, the set of test cases can be run every time a change is made to the code base. **Unit tests** check that small bits of code are correctly implemented. **Functional tests** check that larger chunks of code work correctly. Running the tests can help to identify situations where a change in code in one place breaks the correct operation of some other code. We won't see that advantage of test cases in this textbook, but keep in mind that this introduction to test cases is setting the stage for an essential software engineering practice if you are participating in a larger software development project.

Now it's time to learn how to write code for test cases.

Python provides a statement called `assert`.

- Following the word assert there will be a python expression.
- If that expression evaluates to the Boolean `False`, then the interpreter will raise a runtime error.
- If the expression evaluates to `True`, then nothing happens and the execution goes on to the next line of code.

Why would you ever want to write a line of code that can never compute anything useful for you, but sometimes causes a runtime error? For all the reasons we described above about the value of automated tests. You want a test that will alert that you that some condition you assumed was true is not in fact true. It's much better to be alerted to that fact right away than to have some unexpected result much later in your program execution, which you will have trouble tracing to the place where you had an error in your code.

Why doesn't `assert` print out something saying that the test passed? The reason is that you don't want to clutter up your output window with the results of automated tests that pass. You just want to know when one of your tests fails. In larger projects, other testing harnesses are used instead of `assert`, such as the python `unittest` module. Those provide some output summarizing tests that have passed as well as those that failed. In this textbook, we will just use simple `assert` statements for automated tests.

To write a test, we must know what we *expect* some value to be at a particular point in the program's execution. In the rest of the chapter, we'll see some examples of `assert` statements and ideas for what kinds of assertions one might want to add in one's programs.

**Note**

A note to instructors: this chapter is deliberately structured so that you can introduce testing early in the course if you want to. You will need to cover chapter 8, on Conditionals, before starting this chapter, because that chapter covers Booleans. The subchapters on testing types and testing conditionals can be covered right after that. The subchapter on testing functions can be delayed until after you have covered function definition.

**Check your understanding**

test-1-1: When `assert x==y` is executed and x and y have different values, what will happen?

- (•) A. A runtime error will occur
- ( ) B. A message is printed out saying that the test failed.
- ( ) C. x will get the value that y currently has
- ( ) D. Nothing will happen
- ( ) E. A message is printed out saying that the test passed.

Check me Compare me

✔ The expression ``x==y`` evaluates to ``False``, so a runtime error will occur

Activity: 1 -- Multiple Choice (question19_1_1)

test-1-2: When `assert x==y` is executed and x and y have the same values, what will happen?

- ( ) A. A runtime error will occur
- ( ) B. A message is printed out saying that the test failed.
- ( ) C. x will get the value that y currently has
- (•) D. Nothing will happen
- ( ) E. A message is printed out saying that the test passed.

Check me Compare me

✔ The expression ``x==y`` evaluates to ``True``

Activity: 2 -- Multiple Choice (question19_1_1b)

test-1-3: Test cases are a waste of time, because the python interpreter will give an error message when the program runs incorrectly, and that's all you need for debugging.

- ( ) A. True
- (•) B. False

Check me Compare me

✔ Test cases let you test some pieces of code as you write them, rather than waiting for problems to show themselves later.

Activity: 3 -- Multiple Choice (question19_1_2)

You have attempted 4 of 3 activities on this page

18. Test Cases">

: Cases">

18.2. Checking Assumptions About Data Types">

✔ Comp

18.2. Checking Assumptions About Data Types">Next Section - 18.2. Checking Assumptions About Data Types



| Back to top