PyTest Assertions

What is Assertion?

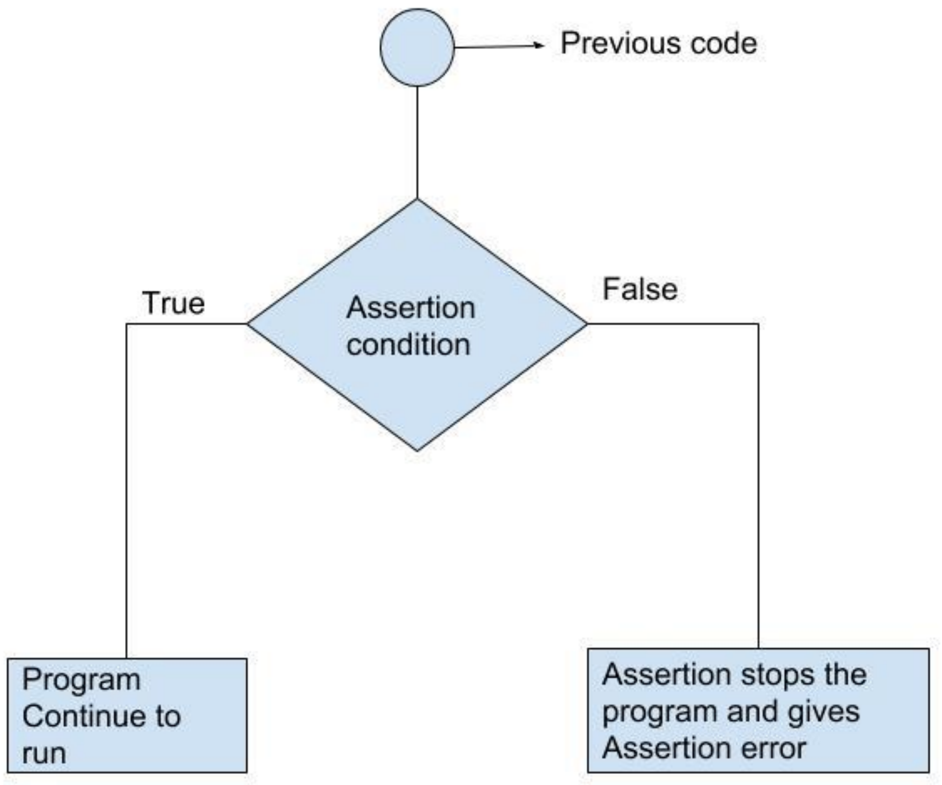
Assertions are statements that assert or state a fact confidently in your program.

For example, while writing a division function, you're confident the divisor shouldn't be zero, you assert divisor is not equal to zero.

Assertions are simply Boolean expressions that checks if the conditions return true or not. If it is true, the program does nothing and move to the next line of code. However, if it's false, the program stops and throws an error.

It is also a debugging tool as it brings the program on halt as soon as any error is occurred and shows on which point of the program error has occurred.

We can be clear by looking at the flowchart below:



Python assert Statement

Python has built-in assert statement to use assertion condition in the program. Assert statement has a condition or expression which is supposed to be always true. If the condition is false assert halts the program and gives an Assertion error.

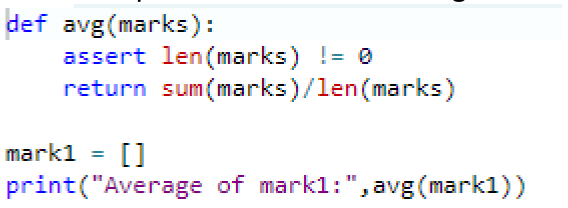
Syntax for using Assert in Python:

```
assert <condition>

assert <condition>, <error message>
```

We can use assert statement in two ways as mentioned above.

1. Assert statement has a condition and if the condition is not satisfied the program will stop and give assertion error.
2. assert statement can also have a condition and an optional error message. If the condition is not satisfied assert stops the program and gives assertion error along with the error message.

Let's take an example, where we have a function which will calculate the average of the values passed by the user and the value should not be an empty list. We will use assert statement to check the parameter and if the length is of the passed list is zero, program halts.



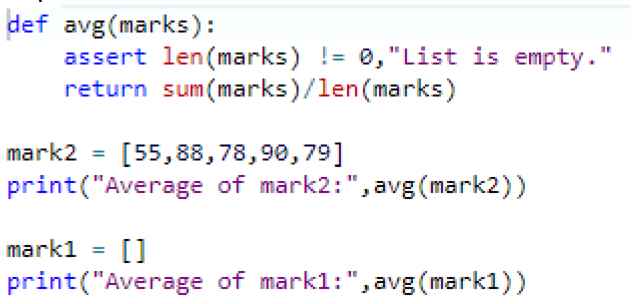
```
def avg(marks):
    assert len(marks) != 0
    return sum(marks)/len(marks)

mark1 = []
print("Average of mark1:",avg(mark1))
```

When we run the above program, the output will be:

```
AssertionError
```

We got an error as we passed an empty list mark1 to assert statement, the condition became false and assert stops the program and give assertion error.

Now let's pass another list which will satisfy the assert condition and see what will be our output.



```
def avg(marks):
    assert len(marks) != 0,"List is empty."
    return sum(marks)/len(marks)

mark2 = [55,88,78,90,79]
print("Average of mark2:",avg(mark2))

mark1 = []
print("Average of mark1:",avg(mark1))
```

When we run the above program, the output will be:

```
Average of mark2: 78.0
AssertionError: List is empty.
```

We passed a non-empty list mark2 and also an empty list mark1 to the avg () function and we got output for mark2 list but after that we got an error assertion error: List is empty. The assert condition was satisfied by the mark2 list and program to continue to run. However, mark1 doesn't satisfy the condition and gives an assertion error.

Key Points to Remember

- Assertions are the condition or Boolean expression which are always supposed to be true in the code.
- assert statement takes an expression and optional message.
- assert statement is used to check types, values of argument and the output of the function.
- assert statement is used as debugging tool as it halts the program at the point where an error occurs.

Why Assertion?

Programming with assertion is a great idea ...

- because they provide ***run-time checks*** of expected results.
- An assert statement simply checks a Boolean condition, and does nothing if it is true but ***immediately terminates the program*** if it is false.
- An assertion allows you to express in code what you ***assume to always be true*** about data at a particular point in execution.
- When you are ***debugging code*** filled with assert statements, ***failures appear earlier*** and closer to the locations of the errors, which make them easier to diagnose and fix.
- assert statements serve as ***test code integrated directly into your implementation***.
- Get in the habit of writing assert statements for conditions that you think would be ***obviously true***.

The writing and reporting of assertions in tests

Asserting with the assert statement

PyTest allows you to use the standard python **"assert"** for verifying expectations and values in Python tests. For example, you can write the following:

```python
# content of test_assert1.py
def f():
    return 3

def test_function():
    assert f() == 4
```

to assert that your function returns a certain value. If this assertion fails you will see the return value of the function call:

```
$ py.test test_assert1.py
======= test session starts ========
platform linux -- Python 3.4.3, pytest-2.8.7, py-1.4.31, pluggy-0.3.1
rootdir: $REGENDOC_TMPDIR, inifile:
collected 1 items

test_assert1.py F

======= FAILURES ========
_______ test_function ________

    def test_function():
>       assert f() == 4
E       assert 3 == 4
E        +  where 3 = f()

test_assert1.py:5: AssertionError
======= 1 failed in 0.12 seconds ========
```

Assertions about expected exceptions

In order to write assertions about raised exceptions, you can use **"pytest.raises"** as a context manager like this:

```python
import pytest

def test_zero_division():
    with pytest.raises(ZeroDivisionError):
        1 / 0
```

Making use of context-sensitive comparisons

PyTest has rich support for providing context-sensitive information when it encounters comparisons. For example:

```python
# content of test_assert2.py

def test_set_comparison():
    set1 = set("1308")
    set2 = set("8035")
    assert set1 == set2
```

if you run this module:

```
$ py.test test_assert2.py
======= test session starts ========
platform linux -- Python 3.4.3, pytest-2.8.7, py-1.4.31, pluggy-0.3.1
rootdir: $REGENDOC_TMPDIR, inifile:
collected 1 items

test_assert2.py F

======= FAILURES ========
_______ test_set_comparison ________

    def test_set_comparison():
        set1 = set("1308")
        set2 = set("8035")
>       assert set1 == set2
E       assert set(['0', '1', '3', '8']) == set(['0', '3', '5', '8'])
E         Extra items in the left set:
E         '1'
E         Extra items in the right set:
E         '5'
E         Use -v to get the full diff

test_assert2.py:5: AssertionError
======= 1 failed in 0.12 seconds ========
```

Special comparisons are done for a number of cases:

- comparing long strings: a context diff is shown
- comparing long sequences: first failing indices
- comparing dicts: different entries