Using Assert and Testing Exceptions

In this lecture I’m going to go over using the built-in Python assert statement in your unit tests and how to test exceptions in Pytest.

Using the assert Statement

```
def test_IntAssert():
    assert 1 == 1
def test_StrAssert():
    assert “str” == “str”
def test_floatAssert():
    assert 1.0 == 1.0
def test_arrayAssert():
    assert [1,2,3] == [1,2,3]
def test_dictAssert():
    assert {“1”:1} == {“1”:1}
```

- Pytest allows the use of the built in python assert statement for performing verifications in a unit test.
- Comparison on all of the python data types can be performed using the standard comparison operators: <, >, <=, >=, ==, and !=
- Pytest expands on the message returned from assert failures to provide more context in the test results.

- Pytest allows the use of the built in python assert statement for performing verifications in a unit test.
- The normal comparison operators can be used on all python data types: less than, greater than, less than or equal, greater than or equal, equal, or not equal
- Pytest expands on the messages that are reported for assert failures to provide more context in the test results.

Comparing Floating Point Values

```
# Failing Test!!!
def test_BadFloatCompare():
    assert (0.1 + 0.2) == 0.3

# Passing Test!!!
def test_GoodFloatCompare():
    val = 0.1 + 0.2
    assert val == approx(0.3)
```

- Validating floating point values can sometimes be difficult as internally the value is a binary fractions (i.e. 1/3 is internally 0.33333333…)
- Because of this some floating point comparisons that would be expected to pass fail.
- The pytest “approx” function can be used to verify that two floating point values are “approximately” equivalent to each other with a default tolerance of 1e-6.

- Validating floating point values can sometimes be difficult as internally the value is stored as a series of binary fractions.
- Because of this some comparisons that we’d expect to pass will fail.
- Pytest provides the “approx” function which will validate that two floating point values are “approximately” the same value as each other to within a default tolerance of 1 time E to the -6 value.

Verifying Exceptions

```
def test_Exception():
    with raises(ValueError)
        raise ValueError
```

- In some cases we want verify that a function throws an exception under certain conditions.
- Pytest provides the “raises” helper to perform this verification using the “with” keyword.
- If the specified exception is not raised in the code block specified after the “raises” line then the test fails.

- In some test cases we need to verify that a function raises an exception under certain conditions.
- Pytest provides the raises helper to perform this verification using the “with” keyword.
- When the “raises” helper is used the unit test will fail if the specified exception is not thrown in the code block after the “raises line.