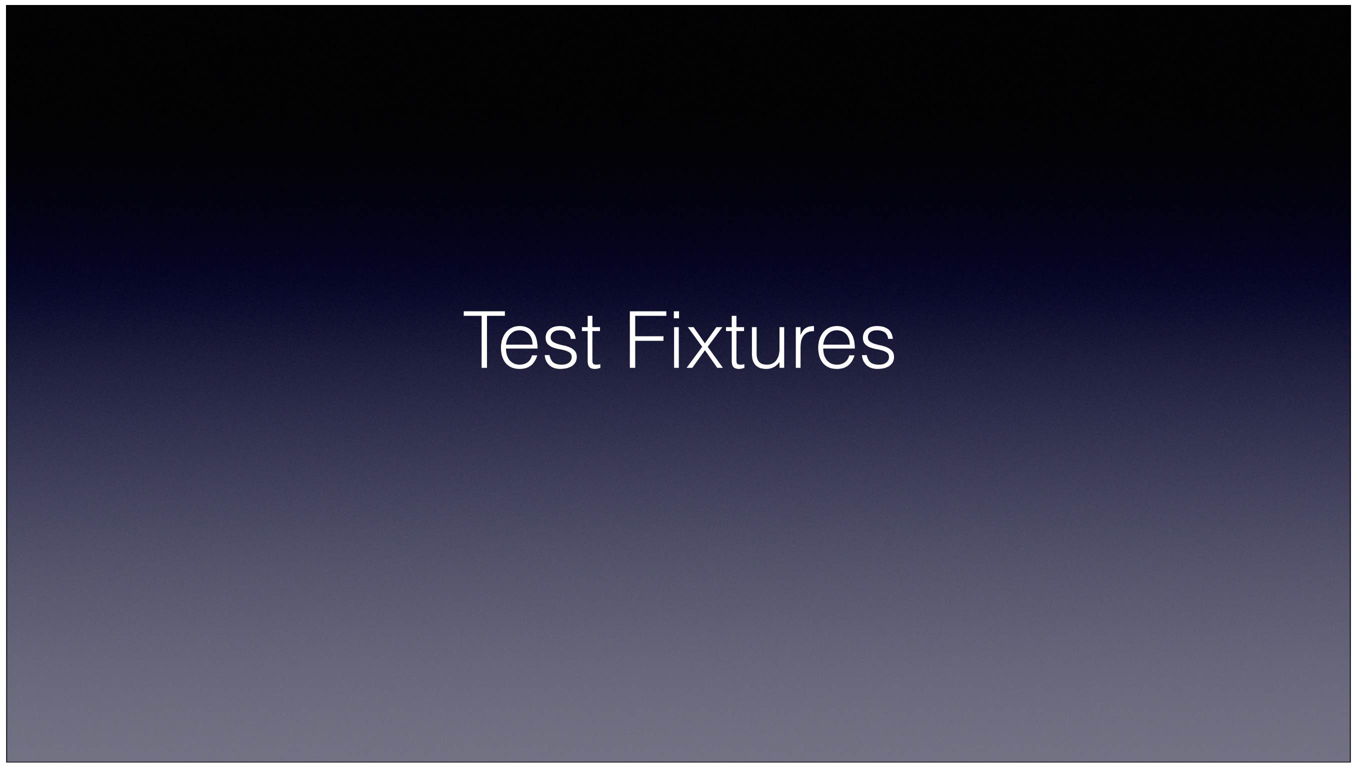
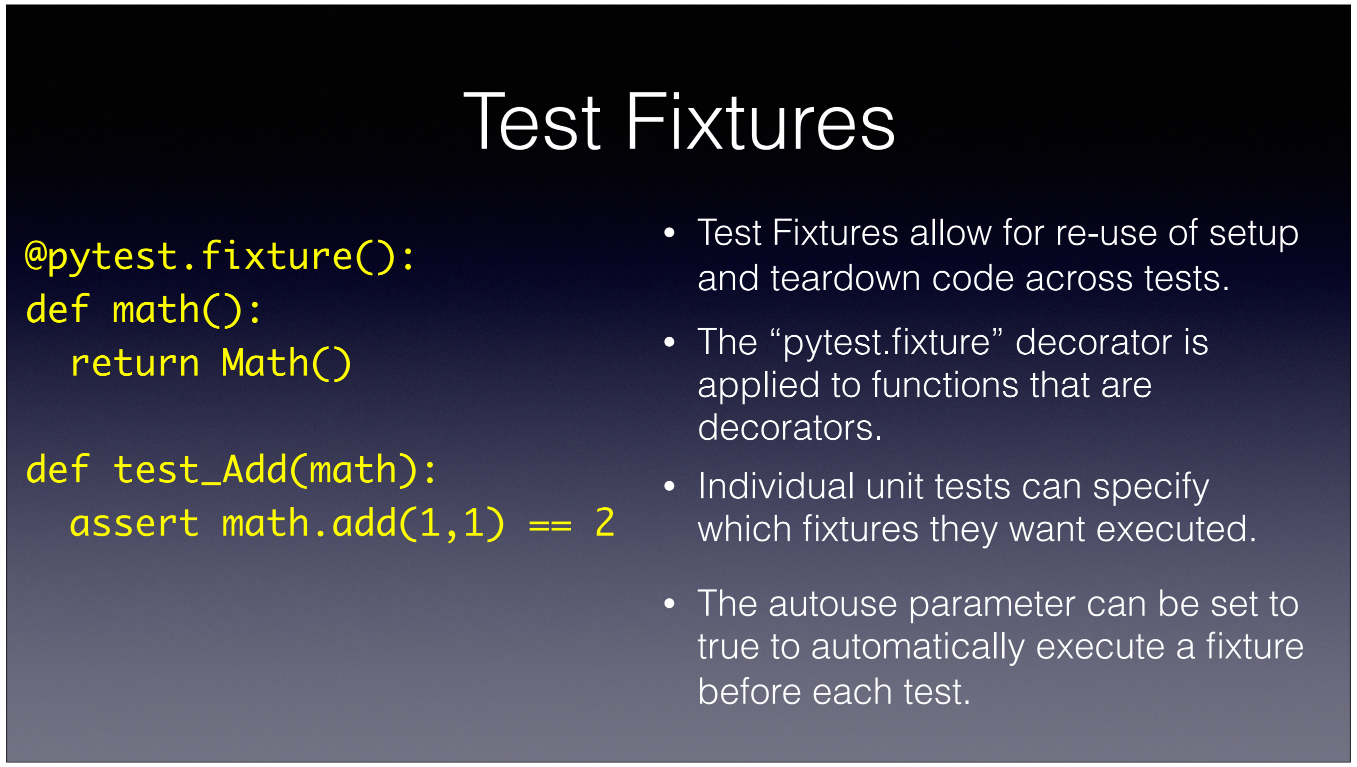


Test Fixtures

In this lecture I’m going to give you an overview of PyTest Fixtures. I’ll explain how Test Fixtures differ from the classic XUnit style setup and teardown functions and I’ll go over some examples.



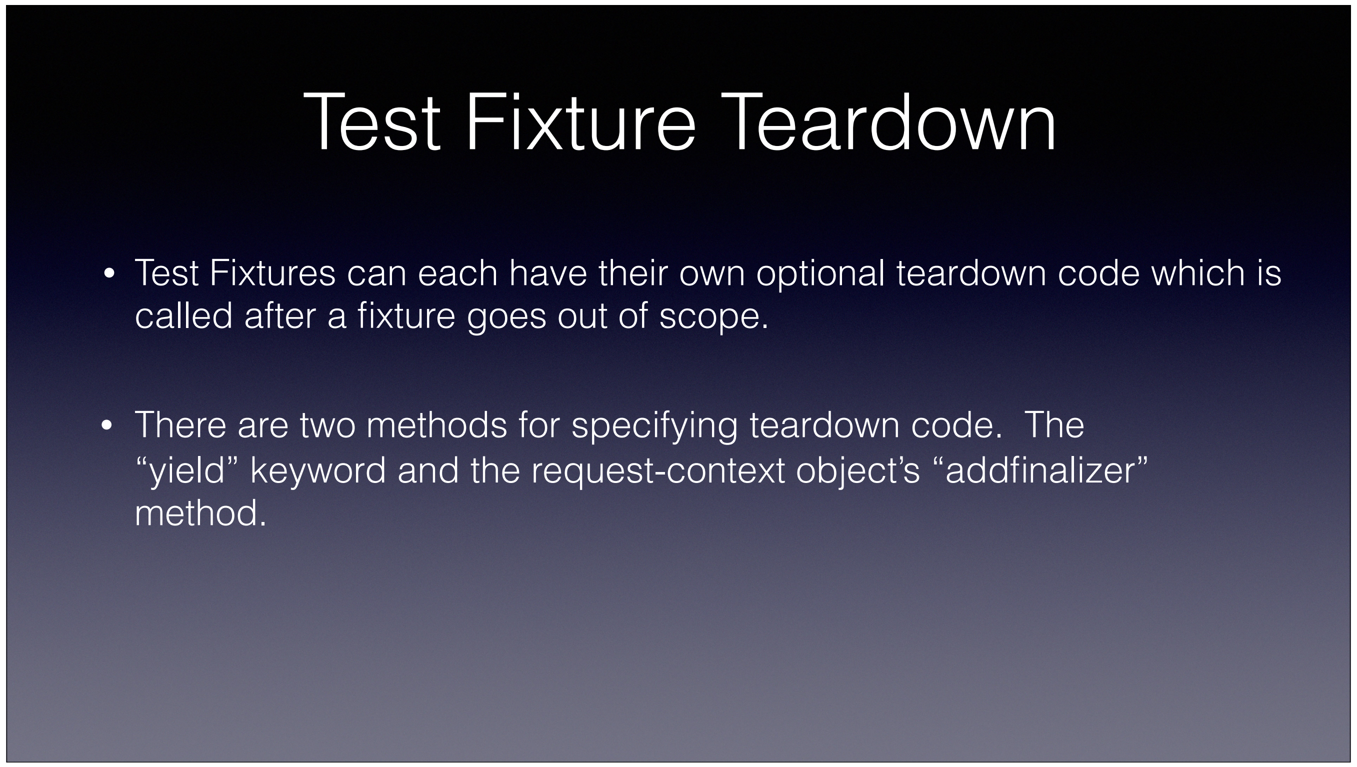
Test Fixtures

```
@pytest.fixture():
def math():
  return Math()

def test_Add(math):
  assert math.add(1,1) == 2
```

- Test Fixtures allow for re-use of setup and teardown code across tests.
- The “pytest.fixture” decorator is applied to functions that are decorators.
- Individual unit tests can specify which fixtures they want executed.
- The autouse parameter can be set to true to automatically execute a fixture before each test.

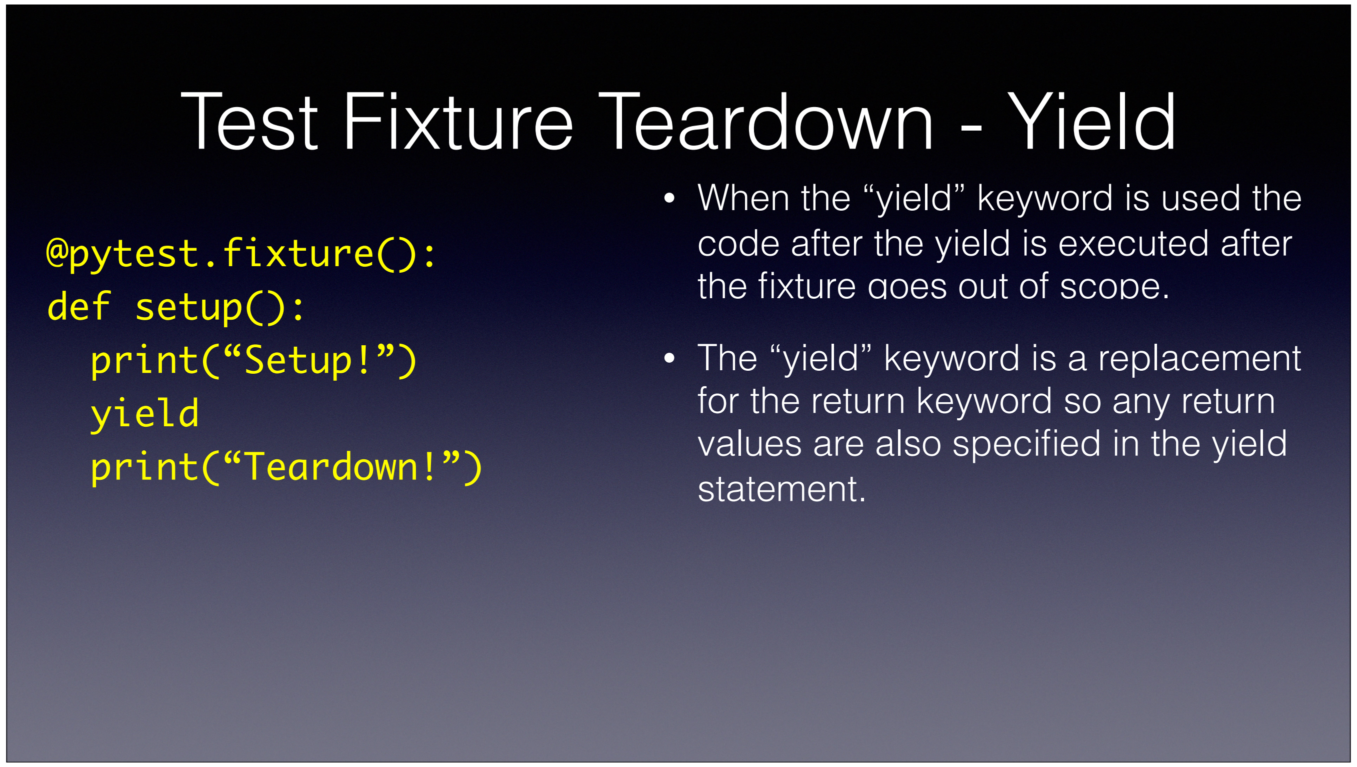
- Like the XUnit style of setup and teardown functions, Test fixtures allow for re-use of code across tests by specifying functions that should be executed before the unit test runs.
- Specifying that a function is a test fixture is done by applying the “pytest.fixture” decorator to the function.
- Individual unit tests can specify they want to use that function by specifying it in their parameter list or by using the “pytest.mark.usefixture” decorator.
- The fixture can also set its autouse parameter to true which will cause all tests in the fixture’s scope to automatically execute the fixture before the test executes.
- Lets look at an example.



Test Fixture Teardown

- Test Fixtures can each have their own optional teardown code which is called after a fixture goes out of scope.
- There are two methods for specifying teardown code. The "yield" keyword and the request-context object's "addfinalizer" method.

- Often there is some type of teardown or cleanup that a test, class, or module need to perform after testing has been completed.
- Each test fixture can specify their own teardown code that should be executed.
- There are two methods of specifying a teardown code for a test fixture: The "yield" keyword and the request-context object's "addfinalizer" method.

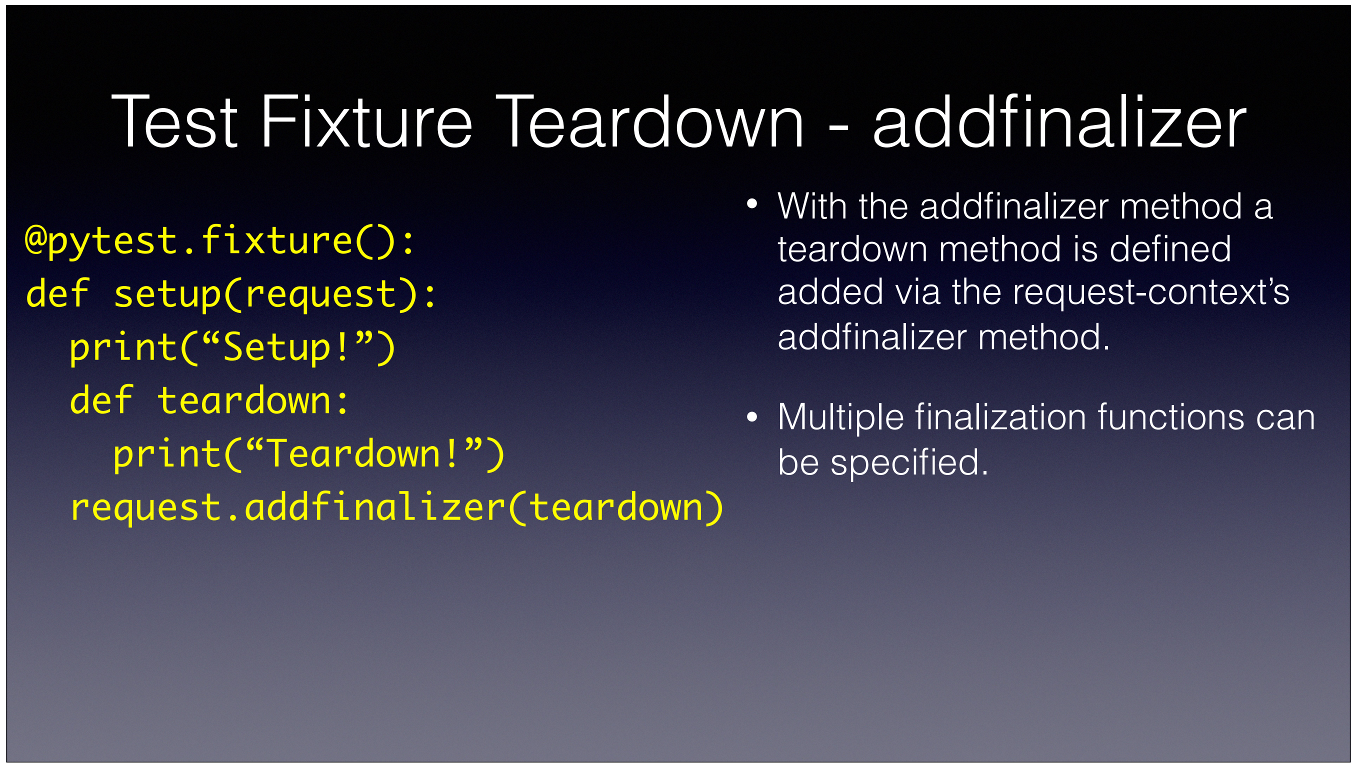


Test Fixture Teardown - Yield

```
@pytest.fixture():
def setup():
  print(“Setup!”)
  yield
  print(“Teardown!”)
```

- When the “yield” keyword is used the code after the yield is executed after the fixture goes out of scope.
- The “yield” keyword is a replacement for the return keyword so any return values are also specified in the yield statement.

- The yield keyword is the simpler of the two options for teardown code.
- The code after the yield is executed after the fixture goes out of scope.
- The yield keyword is a replacement for return and any return values should be passed to it.

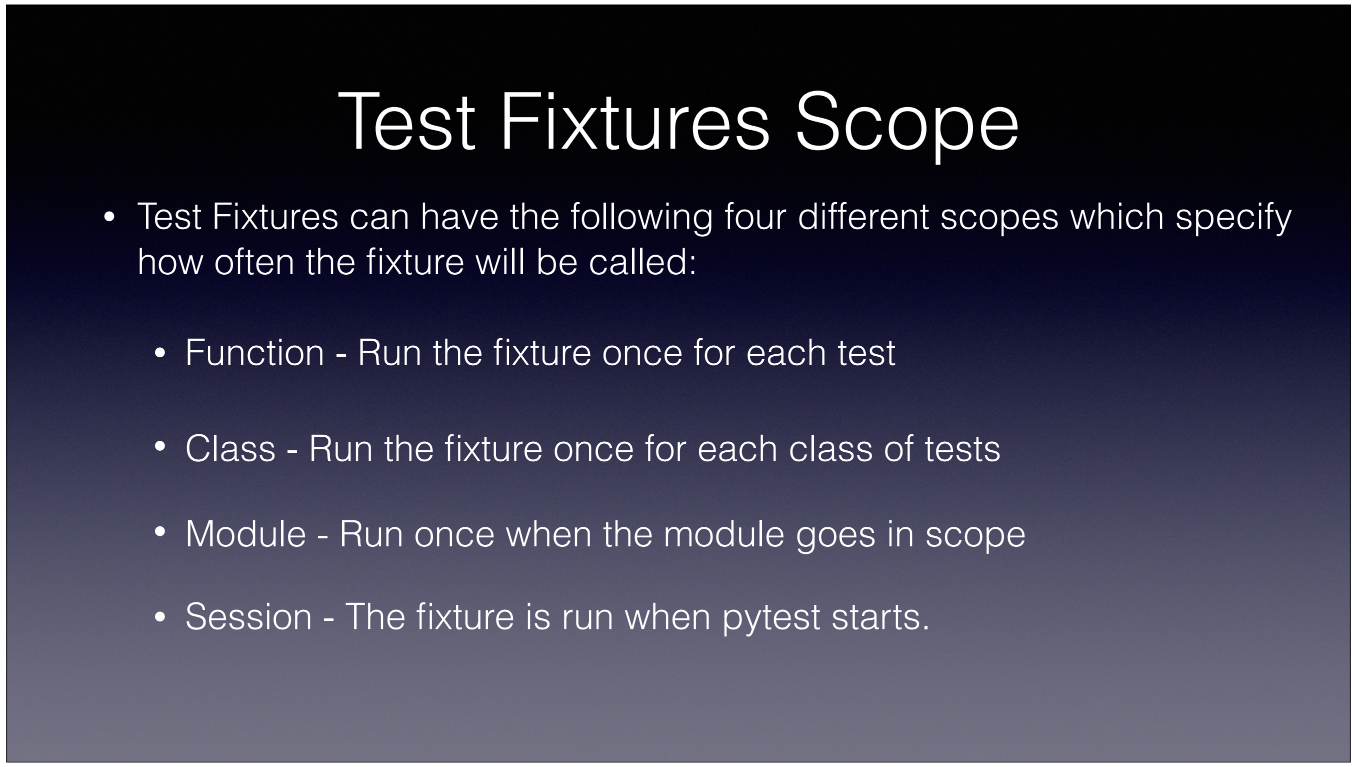


Test Fixture Teardown - addfinalizer

```
@pytest.fixture():
def setup(request):
  print(“Setup!”)
  def teardown:
    print(“Teardown!”)
  request.addfinalizer(teardown)
```

- With the addfinalizer method a teardown method is defined added via the request-context’s addfinalizer method.
- Multiple finalization functions can be specified.

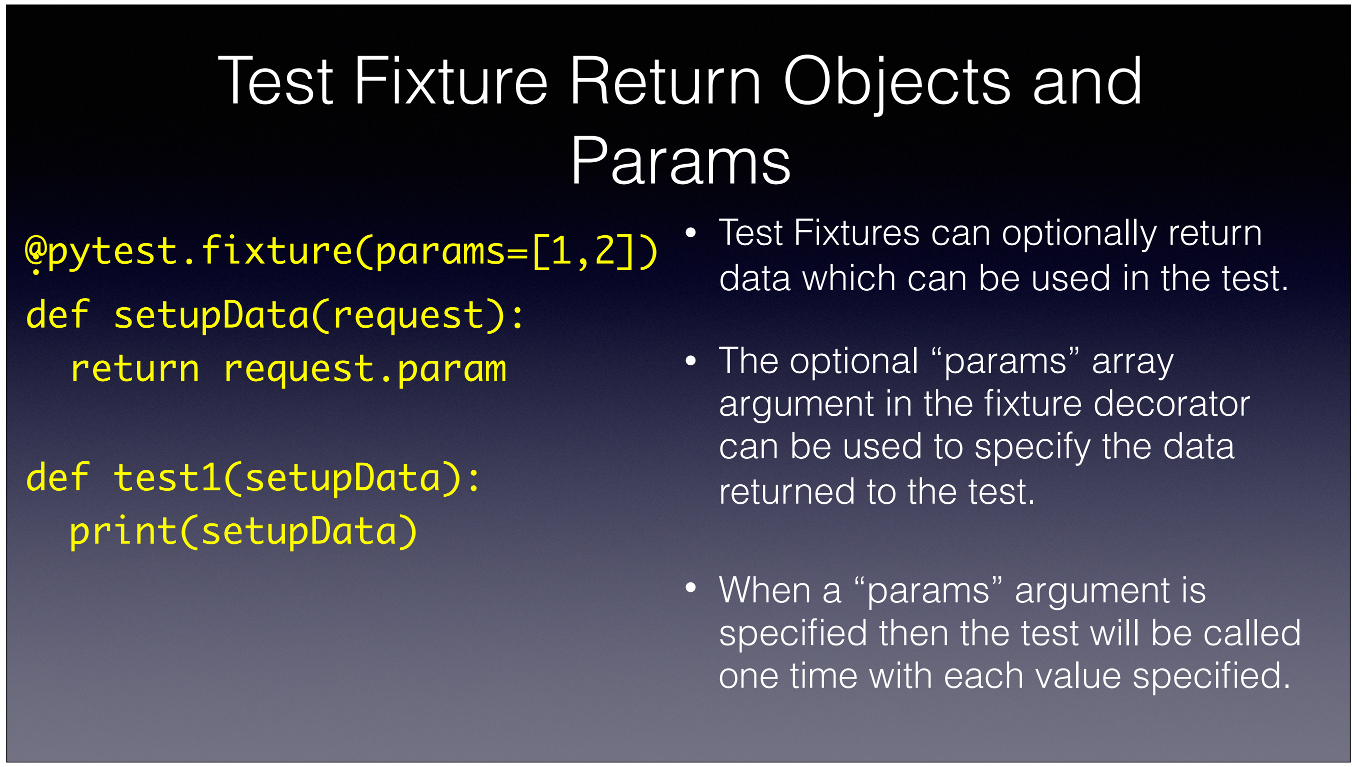
- The addfinalizer method of adding teardown code is a little more complicated but also a little more capable than the yield statement.
- With the addfinalizer method one or more finalizer functions are added via the request-context’s addfinalizer method.
- One of the big differences between this method and the yield keyword method is that this method allows for multiple finalization functions to be specified.
- Now lets take a look at some examples.



Test Fixtures Scope

- Test Fixtures can have the following four different scopes which specify how often the fixture will be called:
  - Function - Run the fixture once for each test
  - Class - Run the fixture once for each class of tests
  - Module - Run once when the module goes in scope
  - Session - The fixture is run when pytest starts.

- Which tests a test fixture applies to and how often it is run depends on the fixture's scope.
- Test fixtures have four different scopes:
- By default the scope is set to function and this specifies that the fixture should be called for all tests in the module.
- Class scope specifies the test fixture should be executed once per test class.
- Module scope specifies that the fixture should be executed once per module.
- Session scope specifies that the fixture should be executed once when PyTest starts.
- Now I'll go over a quick example.



Test Fixture Return Objects and Params

```
@pytest.fixture(params=[1,2])
def setupData(request):
  return request.param

def test1(setupData):
  print(setupData)
```

- Test Fixtures can optionally return data which can be used in the test.
- The optional “params” array argument in the fixture decorator can be used to specify the data returned to the test.
- When a “params” argument is specified then the test will be called one time with each value specified.

- PyTest Test Fixtures allow you to optionally return data from the fixture that can be used in the test.
- The optional params array argument in the fixture decorator can be used to specify one or more values that should be passed to the test.
- When a params argument has multiple values then the test will be called once with each value.
- Lets look at a working example.