Examples README

This README is used to describe the contents of the folder and a walkthrough example.

Folders

Folder	Description
walkthrough	Empty folder to fill out with walkthrough below
script-unit-tests	Contains example as a script-based unit test
function-unit-tests	Contains example as function-based unit test
class-unit-tests	Contains example as class-based unit test
exercise	Simple exercise for writing a function unit tests

The walkthrough example is implemented with script, function, and class based unit tests. The final result of the walkthrough tutorial is located in walkthrough/final.

The exercise folder provides a problem that can be used as an exercise for writing a function and its unit tests.

Getting Started

1. Open MATLAB
2. Change directory to the 'walkthrough' folder

Walkthrough

An easy example is create a function that will sum numeric inputs, `sumNumbers`:

- output is sum of inputs
- accepts zero or more inputs

First, let's create our unit tests to match our description, `testSumNumbers`. The following examples will make use of the function-based unit test.

Create Unit Test File

1. Create file

```
function tests = testSumNumbers
    tests = functiontests(localfunctions);
end
```

> The `function tests = ...` block is necessary for all function-based unit tests. Each test inside the file must begin with `test`.

2. Write tests

- test for no inputs

```
function testNoInput(testCase)
    actSol = sumNumbers1;
    expSol = 0;
    testCase.verifyEqual(actSol, expSol);
end
```

> The `testCase` is a required object used for verification, assumable, and assertable qualifications. See more about this.

- test with one input

```
function testOneInput(testCase)
    actSol = sumNumbers1(1);
    expSol = 1;
    testCase.verifyEqual(actSol, expSol);
end
```

- test with multiple inputs

```
function testMultipleInput(testCase)
    actSol = sumNumbers1(1,2);
    expSol = 3;
    testCase.verifyEqual(actSol, expSol);
end
```

- test with non-numeric input

```
function testNonNumericInput(testCase)
    testCase.verifyError(@() sumNumbers1(''), 'sumNumbers:MustBeNumeric');
end
```

- test with non-scalar input

```
function testNonScalarInput(testCase)
    testCase.verifyError(@() sumNumbers1([2, 3]),
'sumNumbers:MustBeScalar');
end
```

The final file should include the following:

```
function tests = testSumNumbers
    tests = functiontests(localfunctions);
end

function testNoInput(testCase)
    actSol = sumNumbers1;
    expSol = 0;
    testCase.verifyEqual(actSol, expSol);
end

function testOneInput(testCase)
    actSol = sumNumbers1(1);
    expSol = 1;
    testCase.verifyEqual(actSol, expSol);
end

function testMultipleInput(testCase)
    actSol = sumNumbers1(1,2);
    expSol = 3;
    testCase.verifyEqual(actSol, expSol);
end

function testNonNumericInput(testCase)
    testCase.verifyError(@() sumNumbers1(''), 'sumNumbers:MustBeNumeric');
end

function testNonScalarInput(testCase)
    testCase.verifyError(@() sumNumbers1([2, 3]), 'sumNumbers:MustBeScalar');
end
```

Write the function

Write the `sumNumbers` function

```
function s = sumNumbers(varargin)
    %SUMNUMBERS Sum numbers
    %
    % Examples:
    %
    %   s = sumNumbers(0)
    %   s = sumNumbers(1,2,3)

    nargs = length(varargin);
    s = 0;
    for inum = 1:nargs
        % assert(isnumeric(varargin{inum}), 'sumNumbers:MustBeNumeric', 'Inputs
must be numeric');
        s = s + varargin{inum};
    end
end
```

Run Tests

Run the following in the command window:

```
results = run(testSumNumbers)
```

The results will produce

```
>> run(testSumNumbers)
Running testSumNumbers
...
================================================================================
Verification failed in testSumNumbers/testNonNumericInput.
    ----------------------
    Framework Diagnostic:
    ----------------------
    verifyError failed.
    --> The function did not throw any exception.

        Expected Exception:
            'sumNumbers:MustBeNumeric'

    Evaluated Function:
      function_handle with value:

        @()sumNumbers('')
    ------------------
    Stack Information:
    ------------------
    In C:\Users\Trevor\github\matlab-unit-testing\examples\testSumNumbers.m
(testNonNumericInput) at 24
================================================================================
.
================================================================================
Verification failed in testSumNumbers/testNonScalarInput.
    ----------------------
    Framework Diagnostic:
    ----------------------
    verifyError failed.
    --> The function did not throw any exception.

        Expected Exception:
            'sumNumbers:MustBeScalar'

    Evaluated Function:
      function_handle with value:

        @()sumNumbers([2,3])
    ------------------
    Stack Information:
    ------------------
    In C:\Users\Trevor\github\matlab-unit-testing\examples\testSumNumbers.m
```

```
(testNonScalarInput) at 28
================================================================================
.
Done testSumNumbers
__________

Failure Summary:

     Name                                Failed  Incomplete  Reason(s)

================================================================================
     testSumNumbers/testNonNumericInput    X                 Failed by
verification.
    --------------------------------------------------------------------------------
---
     testSumNumbers/testNonScalarInput     X                 Failed by
verification.

ans =

  1×5 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
   3 Passed, 2 Failed (rerun), 0 Incomplete.
   0.11538 seconds testing time.

>>
```

Fix Failure 1

Two tests failed (`testNonNumericInput` and `testNonScalarInput`). First, we will correct `testNonNumericInput`. This can be fixed by asserting an error when an input argument is not a `numeric`.

Adding the line

```
assert(isnumeric(varargin{inum}), 'sumNumbers:MustBeNumeric', 'Inputs must be numeric');
```

inside the for-loop should resolve this failure and ensure inputs are `numeric`.

The updated script should now be

```
function s = sumNumbers(varargin)
    %SUMNUMBERS Sum numbers
    %
    % Examples:
```

```
    %
    %   s = sumNumbers(0)
    %   s = sumNumbers(1,2,3)

    nargs = length(varargin);
    s = 0;
    for inum = 1:nargs
        assert(isnumeric(varargin{inum}), 'sumNumbers:MustBeNumeric', 'Inputs must
be numeric');
        s = s + varargin{inum};
    end
end
```

with the following test results

```
>> run(testSumNumbers)
Running testSumNumbers
....
================================================================================
Verification failed in testSumNumbers/testNonScalarInput.
    ----------------------
    Framework Diagnostic:
    ----------------------
    verifyError failed.
    --> The function did not throw any exception.

        Expected Exception:
            'sumNumbers:MustBeScalar'

    Evaluated Function:
      function_handle with value:

        @()sumNumbers([2,3])
    ------------------
    Stack Information:
    ------------------
    In C:\Users\Trevor\github\matlab-unit-testing\examples\testSumNumbers.m
(testNonScalarInput) at 28
================================================================================
.
Done testSumNumbers
_____

Failure Summary:

     Name                                Failed  Incomplete  Reason(s)

================================================================================
     testSumNumbers/testNonScalarInput    X                 Failed by
verification.

ans =
```

```
  1×5 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
   4 Passed, 1 Failed (rerun), 0 Incomplete.
   0.068778 seconds testing time.

>>
```

Fix Failure 2

The failure of `testNonScalarInput` can be resolved in a similar manner. We will add

`assert(isscalar(varargin{inum}), 'sumNumbers:MustBeScalar', 'Inputs must be scalar');`

inside the for-loop to check that inputs are `scalar`.

The updated script should now be

```
function s = sumNumbers(varargin)
    %SUMNUMBERS Sum numbers
    %
    % Examples:
    %
    %   s = sumNumbers(0)
    %   s = sumNumbers(1,2,3)

    nargs = length(varargin);
    s = 0;
    for inum = 1:nargs
        assert(isnumeric(varargin{inum}), 'sumNumbers:MustBeNumeric', 'Inputs must
be numeric');
        assert(isscalar(varargin{inum}), 'sumNumbers:MustBeScalar', 'Inputs must
be scalar');
        s = s + varargin{inum};
    end
end
```

with the following test results

```
>> run(testSumNumbers)
Running testSumNumbers
.....
```

```
Done testSumNumbers
__________


ans =

  1×5 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
   5 Passed, 0 Failed, 0 Incomplete.
   0.012351 seconds testing time.

>>
```

Try breaking it

If you were to comment `%` either one of the added `assert` lines and rerun our unit tests, not all the tests pass again.

Another way to 'break' our code would be to change our function's implementation/behavior. For example, if decided that sometime in the future it is more appropiate to return an empty `[]` instead of `0` when no outputs are provided (`s = []`), we would receive the following test results if we rerun our unit tests

```
>> run(testSumNumbers)
Running testSumNumbers

================================================================================
Verification failed in testSumNumbers/testNoInput.
    ---------------------
    Framework Diagnostic:
    ---------------------
    verifyEqual failed.
    --> Sizes do not match.

        Actual size:
             0     0
        Expected size:
             1     1

    Actual Value:
         []
    Expected Value:
         0
    ------------------
```

```
    Stack Information:
    ------------------
    In C:\Users\Trevor\github\matlab-unit-testing\examples\testSumNumbers.m
(testNoInput) at 8
================================================================================
.
================================================================================
Verification failed in testSumNumbers/testOneInput.
    ---------------------
    Framework Diagnostic:
    ---------------------
    verifyEqual failed.
    --> Sizes do not match.

        Actual size:
             0     0
        Expected size:
             1     1

    Actual Value:
         []
    Expected Value:
         1
    ------------------
    Stack Information:
    ------------------
    In C:\Users\Trevor\github\matlab-unit-testing\examples\testSumNumbers.m
(testOneInput) at 14
================================================================================
.
================================================================================
Verification failed in testSumNumbers/testMultipleInput.
    ---------------------
    Framework Diagnostic:
    ---------------------
    verifyEqual failed.
    --> Sizes do not match.

        Actual size:
             0     0
        Expected size:
             1     1

    Actual Value:
         []
    Expected Value:
         3
    ------------------
    Stack Information:
    ------------------
    In C:\Users\Trevor\github\matlab-unit-testing\examples\testSumNumbers.m
(testMultipleInput) at 20
================================================================================
...
```

```
Done testSumNumbers
__________

Failure Summary:

     Name                                Failed  Incomplete  Reason(s)

================================================================================
     testSumNumbers/testNoInput          X                 Failed by verification.
    --------------------------------------------------------------------------------
-
     testSumNumbers/testOneInput         X                 Failed by verification.
    --------------------------------------------------------------------------------
-
     testSumNumbers/testMultipleInput    X                 Failed by verification.

ans =

  1×5 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
   2 Passed, 3 Failed (rerun), 0 Incomplete.
   0.23153 seconds testing time.

>>
```

Now 3 tests have failed...

Hopefully you now see the importance or utility of writing unit tests!

Notes

You may also want to know how to implement `testSumNumbers` script-based unit tests or class-based unit tests.