Part 2: Unit testing activity

Function 1

This is a simple function that calculates the sum of an list with integers or floats.

```python
# A function calculating the total of a list from input
def listSum(_list):
    for i in _list:
        if type(i) != int and type(i) != float:
            return None
        else:
            continue
    return sum(_list)

```

Steps

I started with an empty block of code.

```
======================================================================
FAIL: test_is_sum (__main__.TestSetForTest1)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-9-06d897263cf5>", line 4, in test_is_sum
    self.assertEqual(listSum([1, 2, 3]), 6)
AssertionError: [1, 2, 3] != 6

----------------------------------------------------------------------
Ran 1 test in 0.002s

FAILED (failures=1)
```

1. Create a test to check if the input is a list

Created the function with what I was testing in the first test.

```
======================================================================
FAIL: test_is_sum (__main__.TestSetForTest1)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-13-06d897263cf5>", line 4, in test_is_sum
    self.assertEqual(listSum([1, 2, 3]), 6)
AssertionError: None != 6

----------------------------------------------------------------------
Ran 1 test in 0.002s

FAILED (failures=1)
```

2. Create the target function with one parameter, and return nothing, re-run the test

Return what the first test wanted.

```
test_is_sum (__main__.TestSetForTest1) ... ok

----------------------------------------------------
Ran 1 test in 0.002s

OK
```

3. Return 6

Create a second test and of course the test failed because the first out is hard-coded correct.

```
test_is_sum (__main__.TestSetForTest1) ... ok
test_is_sum_10 (__main__.TestSetForTest1) ... FAIL

======================================================================
FAIL: test_is_sum_10 (__main__.TestSetForTest1)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-24-8daedc4671ed>", line 7, in test_is_sum_10
    self.assertEqual(listSum(range(1, 11)), 55)
AssertionError: 6 != 55

----------------------------------------------------------------------
Ran 2 tests in 0.002s

FAILED (failures=1)
```

4. Create the second test to test another list

Finally implementing the function but just in the most basic way.

```
test_is_sum (__main__.TestSetForTest1) ... ok
test_is_sum_10 (__main__.TestSetForTest1) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.003s

OK
```

5. Modify the function to produce the correct output

Create a third test to test unwanted inputs, like strings in this calculating sum function.

```
test_is_nan (__main__.TestSetForTest1) ... ERROR
test_is_sum (__main__.TestSetForTest1) ... ok
test_is_sum_10 (__main__.TestSetForTest1) ... ok

======================================================================
ERROR: test_is_nan (__main__.TestSetForTest1)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-27-d69d7e5a4e8c>", line 10, in test_is_nan
    self.assertEqual(listSum(['1', '2', '3', '4']), None)
  File "<ipython-input-25-01beb545d8ee>", line 3, in listSum
    return sum(_list)
TypeError: unsupported operand type(s) for +: 'int' and 'str'

----------------------------------------------------------------------
Ran 3 tests in 0.003s

FAILED (errors=1)
```

6. Create the third test for edge case

Lastly, improve on the function with data type checking before proceeding calculation to pass the third test.

```
test_is_nan (__main__.TestSetForTest1) ... ok
test_is_sum (__main__.TestSetForTest1) ... ok
test_is_sum_10 (__main__.TestSetForTest1) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.003s

OK
```

7. Modify the function to check if the elements in the parameter list is integer or float, and re-run the test

Tests

```
class TestSetForTest1(unittest.TestCase):

    def test_is_sum(self):
        self.assertEqual(listSum([1, 2, 3]), 6)

    def test_is_sum_10(self):
        self.assertEqual(listSum(range(1, 11)), 55)

    def test_is_nan(self):
        self.assertEqual(listSum(['1', '2', '3', '4']), None)

unittest.main(argv=['ignored', '-v'], exit=False)
```

Function 2

This is a function calculating the percentage of a given index in the list. A list and an index are parameters. List content must be integer or float.

```
# A function taking a list and an index as parameters, calculating the
percentage of the index in the list, rounding to second decimal
def percentageOfIndex(_list, _index):
    for i in _list:
        if type(i) != int and type(i) != float:
            return None
        else:
            continue
    return round(_list[_index]/sum(_list), 2)
```

Steps

Similarly, I started with an empty block of code.

```
test_percentage (__main__.TestSetForTest2) ... ERROR

======================================================================
ERROR: test_percentage (__main__.TestSetForTest2)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-3-07015c69f8ce>", line 4, in test_percentage
    self.assertEqual(percentageOfIndex(), 0.1)
NameError: name 'percentageOfIndex' is not defined

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (errors=1)
```

1. Create a test to check if the return is the percentage

Then I initialized the function with two parameters the first test wanted.

```
test_percentage (__main__.TestSetForTest2) ... FAIL

======================================================================
FAIL: test_percentage (__main__.TestSetForTest2)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-24-a1d7c28cc4ea>", line 4, in test_percentage
    self.assertEqual(percentageOfIndex(range(1, 25, 2), 7), 0.1)
AssertionError: None != 0.1

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
```

2. Create the target function with two parameters, and return nothing, re-run the test

Return what the first test wanted.

```
test_percentage (__main__.TestSetForTest2) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

3. Modify the function to return 0.1

Create a second test to test a different input and failed due to hard-coded.

```
test_percentage (__main__.TestSetForTest2) ... ok
test_percentage2 (__main__.TestSetForTest2) ... FAIL

======================================================================
FAIL: test_percentage2 (__main__.TestSetForTest2)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-37-30b96686741a>", line 7, in test_percentage2
    self.assertEqual(percentageOfIndex(range(1, 25, 3), 6), 0.21)
AssertionError: 0.1 != 0.21

----------------------------------------------------------------------
Ran 2 tests in 0.003s

FAILED (failures=1)
```

4. Create the second test to test another list and index as input

Now implementing the function to pass the first two tests.

```
test_percentage (__main__.TestSetForTest2) ... ok
test_percentage2 (__main__.TestSetForTest2) ... ok

----------------------------------------------------
Ran 2 tests in 0.002s

OK
```

5. Modify the function to produce the correct output

Since calculating percentage is only possible with numbers as content, the third test is some characters mixed inside.

```
======================================================================
ERROR: test_input_nan (__main__.TestSetForTest2)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-40-386faafa7722>", line 10, in test_input_nan
    self.assertEqual(percentageOfIndex(['a', '2', 'q', 0, 4, '0'], 3), None)
  File "<ipython-input-38-0c6cbfaa2140>", line 3, in percentageOfIndex
    return round(_list[_index]/sum(_list), 2)
TypeError: unsupported operand type(s) for +: 'int' and 'str'

----------------------------------------------------------------------
Ran 3 tests in 0.003s

FAILED (errors=1)
```

6. Create the third test for edge case

Improve the function to ensure the data type is either integer or float before calculating.

```
test_input_nan (__main__.TestSetForTest2) ... ok
test_percentage (__main__.TestSetForTest2) ... ok
test_percentage2 (__main__.TestSetForTest2) ... ok

----------------------------------------------------
Ran 3 tests in 0.003s

OK
```

7. Modify the function to check if the elements in the parameter list is integer or float, and re-run the test

Tests

```python
class TestSetForTest2(unittest.TestCase):

    def test_percentage(self):
        self.assertEqual(percentageOfIndex(range(1, 25, 2), 7), 0.1)

    def test_percentage2(self):
        self.assertEqual(percentageOfIndex(range(1, 25, 3), 6), 0.21)

    def test_input_nan(self):
        self.assertEqual(percentageOfIndex(['a', '2', 'q', 0, 4, '0'], 3),
        None)

unittest.main(argv=['ignored', '-v'], exit=False)
```

Function 3

This is a function calculating the mode, which is the most frequent element(s) in a list. The list can be numeric or string/character, or a mixed of them.

```python
# A function calculating the mode, also known as the most frequent value, in a given range

def mostFrequent(_list):
    maxCount = 0
    num = _list[0]
    numList = []

    for i in _list:
        count = _list.count(i)
        if (count > maxCount):
            maxCount = count
            num = i

    for i in _list:
        count = _list.count(i)
        if (count == maxCount):
            numList.append(i)

    numList = sorted(list(set(numList)))
    if (len(numList) == 1):
        return numList[0]
    else:
        return numList
```

Steps

The first step is still an empty block of code with the list input I need as the test.

```
test_frequent (__main__.TestSetForTest3) ... ERROR

======================================================================
ERROR: test_frequent (__main__.TestSetForTest3)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-3-94b589cda574>", line 5, in test_frequent
    self.assertEqual(mostFrequent(testList), 6)
NameError: name 'mostFrequent' is not defined

----------------------------------------------------------------------
Ran 1 test in 0.002s

FAILED (errors=1)
```

1. Create a test to check if the return is the most frequent value

Initialized the function with a parameter.

```
FAIL: test_frequent (__main__.TestSetForTest3)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-5-94b589cda574>", line 5, in test_frequent
    self.assertEqual(mostFrequent(testList), 6)
AssertionError: None != 6

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
```

2. Create the target function with one parameter, and return nothing, re-run the test

Return the most frequent element in test 1 to pass the test.

```
Ran 1 test in 0.002s

OK
```

3. Modify the function to return 6

Have a second test for a different input.

```
FAIL: test_frequent2 (__main__.TestSetForTest3)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-8-4a45e664e899>", line 9, in test_frequent2
    self.assertEqual(mostFrequent(testList), 8)
AssertionError: 6 != 8

----------------------------------------------------------------------
Ran 2 tests in 0.004s

FAILED (failures=1)
```

4. Create the second test to test another list as input

Implement the function to actually return whichever is the most frequent element.

```
test_frequent (__main__.TestSetForTest3) ... ok
test_frequent2 (__main__.TestSetForTest3) ... ok

----------------------------------------------------
Ran 2 tests in 0.002s

OK
```

5. Modify the function to produce the correct output

There could be more edge cases for this one so the third test combined two of them. Having mixed input data types as well as having two or more most frequent elements.

```
FAIL: test_input_mixed (__main__.TestSetForTest3)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-15-7da9e4e27bc1>", line 13, in test_input_mixed
    self.assertEqual(mostFrequent(testList), [4, 5])
AssertionError: 5 != [4, 5]

----------------------------------------------------------------------
Ran 3 tests in 0.005s

FAILED (failures=1)
```

6. Create the third test for edge case (mixed number and string, two most frequent values)

Improve the function to take the two edge cases into considerations.

```
test_frequent (__main__.TestSetForTest3) ... ok
test_frequent2 (__main__.TestSetForTest3) ... ok
test_input_mixed (__main__.TestSetForTest3) ... ok

----------------------------------------------------
Ran 3 tests in 0.003s

OK
```

7. Modify the function to allow multiple most frequent values, and re-run the test

Tests

```python
class TestSetForTest3(unittest.TestCase):

    def test_frequent(self):
        testList = ([3]*5) + ([2]*4) + ([6]*7) + ([5]*2)
        self.assertEqual(mostFrequent(testList), 6)

    def test_frequent2(self):
        testList = ([8]*5) + ([4]*4) + ([6]*2) + ([1]*2)
        self.assertEqual(mostFrequent(testList), 8)

    def test_input_mixed(self):
        testList = (['t']*2) + ([5]*9) + (['0']*7) + ([4]*9)
        self.assertEqual(mostFrequent(testList), [4, 5])

unittest.main(argv=['ignored', '-v'], exit=False)

```