Part 1: Module Coupling and Cohesion

Program Purpose

This program's purpose is to convert a static web application about Chess into a dynamic one with interactive web content. The program adds behavior to the web application by enabling the user to interact with its content. The program improves accessibility via sticky navigation bars, prevents element collisions by dynamically adjusting the visibility of elements, creates an engaging UI via slide-in and fade-inanimations, and creates interactive elements such as a carousel containing news article headers.

Technologies Used

The web application uses Javascript, a high level, just-in-time (JIT) compiled language that is native to the web browser. The program also incorporates the jQuery library, which is a feature rich, open-source JS library which simplifies DOM tree traversal, event handling, Ajax and CSS animation. The code extracts selected are written in vanilla JS.

Module Coupling Example 1

The purpose of the following two modules is to determine whether the current page is the 'Current News' page. If the page is confirmed to be the 'Current News' page, then the social media icons aside should be hidden when the user scrolls the page near the footer. This social media icon visibility toggling prevents collisions with the footer section.

Module 1: This module determines if the current page is the 'Current News' page.

```
function isCurrentNewsPage() {
  return (document.querySelector('.fixed-social-media-icons') != null);
}
```

Module 2: This module hides the social media icons at a certain vertical page offset to prevent collisions with the web page's footer. The social media icon visibility is only toggled if the current page is the 'Current News' page.

```
function toggleSocialIconVisibility(isCurrentNewsPage) {
  //  Retrieve the social media icons aside from the DOM
  const fixedSocialMediaAside = document.querySelector('.fixed-social-media-icons');
  /*  Record the starting vertical scroll position (position of 0 is not used or else unexpected behavior occurs when
the page is scrolled to the top */
  let pastScrollPosition = -1;
  //  Use another module to determine if the social icons should be toggled
  let isCurrentNewsFlag = isCurrentNewsPage();
  // Show/hide social media aside based on changes in the user's scroll position for the 'Current News' page only
  if (toggleIconsFlag) {
    document.addEventListener('scroll', (event) => {
      let currentScrollPosition = window.pageYOffset || document.documentElement.scrollTop;
      // Hide the social media icons aside to prevent collision with the footer section
```

```
        if (currentScrollPosition > pastScrollPosition && window.pageYOffset > 5200) {
            fixedSocialMediaAside.style.display = 'none';
        }
        else {
            // Make the aside visible once collision with the footer will not occur
            fixedSocialMediaAside.style.display = 'block';
        }
    })
  }
}
```

This code extract uses **control coupling** between modules, in which data is passed that controls the flow of another module. The first module serves as a flag that determines whether the current page contains the social media icons. This flag is passed as a parameter to the second module. This creates an interdependence in which the first module changes the execution path of the second module by influencing its internal logic. If the flag is true, the social media icons are toggled based on the page's vertical scroll position relative to a threshold, and if it evaluates to false, the toggling logic is not executed. Control coupling tends to be undesirable because it is better to retain the control in a single software component.

Module Coupling Example 2

The purpose of the following two modules is to enable a sticky navigation bar for accessible navigation. The default navigation bar becomes a sticky navigation bar once the user's vertical scroll position has exceeded a threshold value.

Module 1: This module is responsible for retrieving the elements of the DOM that constitute the navigation bar.

```
function retrieveNavigationBar() {
  // Sticky navigation bar allows accessible page navigation without the user having to scroll to the top
  window.addEventListener('scroll', () => {
    // Retrieve the navigation menu from the DOM
    const navBar = document.querySelector('.nav-bar');
    // Retrieve the 'Toggle Contrast' button from the DOM
    const contrastToggle = document.querySelector('.low-contrast-toggle');
    // Retrieve the company logo from the DOM
    const companyLogo = document.querySelector('.company-logo-scroll');
    enableStickyNavigation(navBar, contrastToggle, companyLogo);
  })
}
```

Module 2: This module enables sticky navigation on each of the navigation bar components. Enabling sticky navigation on each component makes the entire navigation bar appear sticky.

```
function enableStickyNavigation(navBar, contrastToggleButton, companyLogo) {
  // Compute the distance from the top of the page
  let offsetTop = navBar.offsetTop;
  // Make navigation menu fixed to the top of the screen when a certain vertical scroll threshold is reached
  navBar.classList.toggle('sticky', window.scrollY > navBar.offsetTop);
  /* Position the 'Contrast Toggle' button relative to the dimensions of the fixed navigation menu when the
  the vertical scroll bar passes a threshold */
  contrastToggle.classList.toggle('sticky-adjust', window.scrollY > navBar.offsetTop);
  /* Display the company logo in a white box in the top left corner of the fixed navigation menu when a
  vertical scroll threshold is reached */
  companyLogo.classList.toggle('company-logo-scroll-active', window.scrollY > navBar.offsetTop);
}
```

This code extract uses **data coupling** between modules, in which the output from one module serves as input to another module. The first module queries the navigation bar components from the DOM and shares the data to the second module via parameter passing. This shared data is composed of a small argument list that is inputted into Module 2, which applies sticky navigation to each component. Since the data is not used to control the flow of the second module, it is not control coupling and therefore an acceptable form of loose coupling. Data coupling is used since the argument list is small, and Module 2 only sees the data elements it requires from Module 1.

Module Cohesion Example 1

The purpose of this module is to enable the user to scroll through the slides of a carousel in a circular manner. When the carousel reaches the last slide, the next slide displayed will be the first slide. Similarly, if a user is on the first carousel slide, the previous slide displayed will be the last slide.

```
function enableCarouselScrolling() {
  // Retrieve the two carousel scrollers from the DOM
  const scrollers = document.querySelectorAll('[data-scroller]');

  // For each carousel scroller, move to the next or previous slide
  scrollers.forEach(scroller => {
    scroller.addEventListener('click', () => {
      // Select all of the carousel slides
      const slides = document.querySelector('[data-slides]');
      // Record the number of carousel slides
      const numSlides = slides.children.length;
      // Determine whether or not to move to next or previous slide
      const offset = scroller.dataset.carouselScroller === 'previous' ? -1 : 1;
      // Retrieve currently visible slide from the DOM
      const visibleSlide = slides.querySelector('[data-visible]');
      // Determine which slide to display by moving forwards or backwards from the currently visible slide
```

```
        let newSlide = offset + [...slides.children].indexOf(visibleSlide);
        // Allow carousel to loop through the slides continuously
        newSlide = newSlide < 0 ? numSlides - 1 : newSlide % numSlides;
        // Make new slide visible
        slides.children[newSlide].dataset.visible = true;
        // Hide the old slide so that a maximum of one slide is displayed at any one instance
        delete visibleSlide.dataset.visible;
      })
    })
}
```

This code extract uses **procedural cohesion**, in which tasks performed by the module all contribute to a given program procedure.

The sequence of execution involves:

1. Determine the next slide to display based on the current visible slide and the click action on either the left or right scroller
2. Make the carousel circular
3. Render the visibility of the next slide
4. Hide all slides that do not include the next slide to be displayed

Thus, procedural cohesion is used, as the parts of the module always follow a procedure, or sequence of execution.

Module Cohesion Example 2
The purpose of this module is to fade in and slide in elements once they are sufficiently in the viewport.

```
function animateDOMElements() {
  // Retrieve all fade-in elements from the DOM
  const fadeInElements = document.querySelectorAll('.fade-in-effect');

  const appearOptions = {
    /* The visibility of an element is determined relative to viewport
    This will be used to determine when the element should be faded in */
    root: null,
    // Fade in occurs once the element has been scrolled into view
    rootMargin: '0px 0px -165px 0px',
    threshold: 0
  };

  // Observer starts to fade in elements that intersect with the viewport
```

```
const intersectionObserver = new IntersectionObserver((entries, intersectionObserver) => {
    entries.forEach(entry => {
        // Apply effect only to elements that overlap with the viewport on page load
        if (entry.isIntersecting) {
            // Make the element visible
            entry.target.classList.add('appear');
            // Ignore the target element after it has been faded in to avoid repeated fade in effects
            intersectionObserver.unobserve(entry.target);
        }
        // Ignore element do not intersect with the viewport on page load
        else {
            return;
        }
    });
},
    appearOptions);

/* Intersection observer determines the relationship of each element to the viewport to determine if it should be
faded in or not */
fadeInElements.forEach(element => {
    intersectionObserver.observe(element);
});

// Add a smooth slide in effect to elements once they are sufficiently visible in the viewport
// Retrieve all slide in elements from the DOM
const slideInElements = document.querySelectorAll('.slide-in-effect');

// Apply a slide in effect to each element
slideInElements.forEach(element => {
    intersectionObserver.observe(element);
})
}
```

This code extract uses **logical cohesion**, in which the tasks performed by the module perform logically similar functions. The module groups together elements that perform an animation based on their intersection with the viewport. Despite both functions contributing to the site's interactivity via dynamic, animated content, they achieve this in different ways. One achieves the animation through a fading in animation that involves a change in opacity and the other through a sliding in animation, which involves a change in the element's coordinates in space. They are categorized together but they are different by nature. Logical cohesion tends to be undesirable,

since it is grouping functionality by the same general, logical categorization as opposed to functional characteristics.

Part 2: Unit Testing Activity - Terrible Fall: C++ Physics Engine

Test Set 1: getMin() Function

Strategy

This test set is used to develop a statistical function used to compute the minimum of a set of numbers in a list. The minimum is the smallest value of the list. TDD is used to incrementally build the getMin() function by writing a failing test and producing the minimal code to pass the test; this process is repeated. The first test tests output for a sorted list of positive and negative integer elements. The second tests output for an unsorted list of positive and negative integer elements. The third tests output for an unsorted list of strictly positive integer elements.

Test Set 1: Test 1

The purpose of this test is to determine if the getMin() function produces the correct output when the list is sorted, and contains positive and negative integer elements. The minimum number is in the first position in the list, as it is sorted by default.

Create a failing test.

```
import unittest
import snakestats

class MinTestForSnakeStats(unittest.TestCase):

    def test_min_sorted_list(self):
        min = snakestats.getMin([-2,-1,0,1,2,3])
        return self.assertEqual(min,-2)

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

Test failed because getMin() does not match any function signature in snakestats.py.

```
test_min_sorted_list (__main__.MinTestForSnakeStats) ... ERROR

======================================================================
ERROR: test_min_sorted_list (__main__.MinTestForSnakeStats)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-1-f923e52fd461>", line 7, in test_min_sorted_list
    min = snakestats.getMin([-2,-1,0,1,2,3])
AttributeError: module 'snakestats' has no attribute 'getMin'

----------------------------------------------------------------------
Ran 1 test in 0.003s

FAILED (errors=1)
```

The minimal code needed to pass this test is to return the first element in the array.

```
1 def getMin(vals):
2     return vals[0]
```

Test successful.

```
test_min_sorted_list (__main__.MinTestForSnakeStats) ... ok

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

OK
```

Test set 1: Test 2

The purpose of this test is to determine if the getMin() function produces the correct output when the input is unsorted and contains positive and negative integers. The index of the minimum element is in an arbitrary position.

Initial code.

```
1 def getMin(vals):
2     return vals[0]
```

Create a failing test.

```
import unittest
import snakestats

class MinTestForSnakeStats(unittest.TestCase):

    def test_min_sorted_list(self):
        min = snakestats.getMin([-2,-1,0,1,2,3])
        return self.assertEqual(min,-2)

    def test_min_unsorted_list(self):
        min = snakestats.getMin([-1,0,2,-2,1,3])
        return self.assertEqual(min,-2)

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

Test failed because getMin() returns the first element. In the unsorted test list, this does not produce the correct output as the minimum element is located at index 3.

```
test_min_sorted_list (__main__.MinTestForSnakeStats) ... ok
test_min_unsorted_list (__main__.MinTestForSnakeStats) ... FAIL

======================================================================
FAIL: test_min_unsorted_list (__main__.MinTestForSnakeStats)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-2-218bb5cecafc>", line 12, in test_min_unsorted_list
    return self.assertEqual(min,-2)
AssertionError: -1 != -2

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

FAILED (failures=1)
```

The minimal code needed to pass this test is to initialize a minimumSeen variable to 0 and iterate over the entire length of the list. For every iteration, the current element is compared to the current minimum and minimumSeen is updated if a lower value is encountered.

```
def getMin(vals):
    minimumSeen = 0
    for i in range(len(vals)):
        if (vals[i] < minimumSeen):
            minimumSeen = vals[i]
    return minimumSeen
```

Test successful.

```
test_min_sorted_list (__main__.MinTestForSnakeStats) ... ok
test_min_unsorted_list (__main__.MinTestForSnakeStats) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

Test set 1: Test 3

The purpose of this test is to determine if the getMin() function produces the correct output for an unsorted list with strictly positive integer values.

Initial code.

```
def getMin(vals):
    minimumSeen = 0
    for i in range(len(vals)):
        if (vals[i] < minimumSeen):
            minimumSeen = vals[i]
    return minimumSeen
```

Create a failing test.

```
import unittest
import snakestats

class MinTestForSnakeStats(unittest.TestCase):

    def test_min_sorted_list(self):
        min = snakestats.getMin([-2,-1,0,1,2,3])
        return self.assertEqual(min,-2)

    def test_min_unsorted_list(self):
        min = snakestats.getMin([-1,0,2,-2,1,3])
        return self.assertEqual(min,-2)

    def test_min_unsorted_positive_integer_list(self):
        min = snakestats.getMin([3,2,6,7,1,4,5])
        return self.assertEqual(min,1)

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

Test failed because getMin() initializes the minimumSeen variable to zero. This assumes that the minimum element is less than zero; in this case, the assumed minimum of 0 is overriden. In a list containing strictly positive numbers, the initial value of minimumSeen will not be overridden and the produced output is incorrect as 0 does not belong to the list.

```
test_min_sorted_list (__main__.MinTestForSnakeStats) ... ok
test_min_unsorted_list (__main__.MinTestForSnakeStats) ... ok
test_min_unsorted_positive_integer_list (__main__.MinTestForSnakeStats) ... FAIL

======================================================================
FAIL: test_min_unsorted_positive_integer_list (__main__.MinTestForSnakeStats)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-1-e9348b3c2951>", line 16, in test_min_unsorted_positive_integer_list
    return self.assertEqual(min,1)
AssertionError: 0 != 1

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

FAILED (failures=1)
```

The minimal code needed to pass this test is to initialize minimumSeen as the value of the first element in the list, and to begin iterating from the second element to the end of the list. This ensures that the minimumSeen variable always holds a valid value of the list.

```
1 def getMin(vals):
2     minimumSeen = vals[0]
3     for i in range(1,len(vals)):
4         if (vals[i] < minimumSeen):
5             minimumSeen = vals[i]
6     return minimumSeen
```

Test successful.

```
test_min_sorted_list (__main__.MinTestForSnakeStats) ... ok
test_min_unsorted_list (__main__.MinTestForSnakeStats) ... ok
test_min_unsorted_positive_integer_list (__main__.MinTestForSnakeStats) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.002s

OK
```

Test Set 2: getMedian() Function

This test set is used to develop a statistical function to compute the median of a set of numbers in a list. The median is the middle number in a sorted, ascending or descending, list of numbers. TDD is used to incrementally build the getMedian() function by writing a failing test and producing the minimal code to pass the test; this process is repeated. The first test tests output for a sorted list of data with an odd number of elements. The second test tests output for an unsorted list of data with an odd number of elements. The third test tests output for an unsorted list of data with an even number of elements.

Test Set 2: Test 1

The purpose of this test is to determine if the getMedian() function produces the correct output when the list is sorted and contains an odd number of elements.

Create a failing test.

```
import unittest
import snakestats

class MedianTestForSnakeStats(unittest.TestCase):

    def test_median_odd_length_sorted(self):
        median = snakestats.getMedian([1,2,3,4,5,6,7])
        return self.assertEqual(median,4)

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

Test failed because getMedian() does not match any function signature in snakestats.py.

```
test_median_odd_length_sorted (__main__.MedianTestForSnakeStats) ... ERROR

======================================================================
ERROR: test_median_odd_length_sorted (__main__.MedianTestForSnakeStats)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-1-9e6988465931>", line 7, in test_median_odd_length_sorted
    median = snakestats.getMedian([1,2,3,4,5,6,7])
AttributeError: module 'snakestats' has no attribute 'getMedian'

----------------------------------------------------------------------
Ran 1 test in 0.003s

FAILED (errors=1)
```

The minimal code needed to pass this test is to determine the length of the list. The index of the median element is computed by performing a floor division on the length by two. The element in the list at this index is returned.

```
1  def getMedian(vals):
2      length = len(vals)
3      return vals[length//2]
```

Test successful.

```
test_median_odd_length_sorted (__main__.MedianTestForSnakeStats) ... ok

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

OK
```

Test Set 2: Test 2

The purpose of this test is to determine if the getMedian() function produces the correct output when the list is unsorted and contains an odd number of elements.

Initial code.

```
1  def getMedian(vals):
2      length = len(vals)
3      return vals[length//2]
```

Create a failing test.

```
import unittest
import snakestats

class MedianTestForSnakeStats(unittest.TestCase):

    def test_median_odd_length_sorted(self):
        median = snakestats.getMedian([1,2,3,4,5,6,7])
        return self.assertEqual(median,4)

    def test_median_odd_length_unsorted(self):
        median = snakestats.getMedian([4,3,2,5,7,1,6])
        return self.assertEqual(median,4)

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

Test failed because getMedian() returns the median element assuming that the initial list is sorted. In an unsorted list, the median element is not necessarily positioned at the index given by the floor division of the length by two.

```
test_median_odd_length_sorted (__main__.MedianTestForSnakeStats) ... ok
test_median_odd_length_unsorted (__main__.MedianTestForSnakeStats) ... FAIL

======================================================================
FAIL: test_median_odd_length_unsorted (__main__.MedianTestForSnakeStats)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-1-e59c2f03785f>", line 12, in test_median_odd_length_unsorted
    return self.assertEqual(median,4)
AssertionError: 5 != 4

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

FAILED (failures=1)
```

The minimal code needed to pass this test is to sort the list prior to returning the median of the list at the index, which is obtained by performing a floor division by two on the length of the list. Since the list is of even length, there is only one middle element.

```
1 def getMedian(vals):
2     length = len(vals)
3     return sorted(vals)[length//2]
```

Test successful.

```
test_median_odd_length_sorted (__main__.MedianTestForSnakeStats) ... ok
test_median_odd_length_unsorted (__main__.MedianTestForSnakeStats) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

Test Set 2: Test 3

The purpose of this test is to determine if the getMedian() function produces the correct output when the list is unsorted and contains an even number of elements.

Initial code.

```
def getMedian(vals):
    length = len(vals)
    return sorted(vals)[length//2]
```

Create a failing test.

```
import unittest
import snakestats

class MedianTestForSnakeStats(unittest.TestCase):

    def test_median_odd_length_sorted(self):
        median = snakestats.getMedian([1,2,3,4,5,6,7])
        return self.assertEqual(median,4)

    def test_median_odd_length_unsorted(self):
        median = snakestats.getMedian([4,3,2,5,7,1,6])
        return self.assertEqual(median,4)

    def test_median_even_length_unsorted(self):
        median = snakestats.getMedian([4,2,3,5,7,1,8,6])
        return self.assertEqual(median,4.5)

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

Test failed because getMedian() returns the median element assuming a list with an odd number of elements. In a list with an even number of elements, the two middle elements must be averaged by summing them and dividing by two. This even-case handling functionality is not built into getMedian() .

```
test_median_even_length_unsorted (__main__.MedianTestForSnakeStats) ... FAIL
test_median_odd_length_sorted (__main__.MedianTestForSnakeStats) ... ok
test_median_odd_length_unsorted (__main__.MedianTestForSnakeStats) ... ok

======================================================================
FAIL: test_median_even_length_unsorted (__main__.MedianTestForSnakeStats)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-1-a1dfcddf3b18>", line 16, in test_median_even_length_unsorted
    return self.assertEqual(median,4.5)
AssertionError: 5 != 4.5

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

FAILED (failures=1)
```

The minimal code needed to pass this test is to add a condition that determines whether the list is odd or even in length, and a return statement that executes if the length of the list is even. Since the list is of even length, there will be two middle elements. The first middle element has an index of length-1 floor division two. The second middle element has an index of length floor division two. The sum function is applied to a slice ranging from the index of the first middle element to one greater than the index of the second middle element (inclusive to exclusive slice). This slice is performed on the sorted list and divided by two to compute the median in the even case.

```
def getMedian(vals):
    length = len(vals)
    if length%2:
        return sorted(vals)[length//2]
    return sum(sorted(vals)[length//2-1:length//2+1])/2
```

Test successful.

```
test_median_even_length_unsorted (__main__.MedianTestForSnakeStats) ... ok
test_median_odd_length_sorted (__main__.MedianTestForSnakeStats) ... ok
test_median_odd_length_unsorted (__main__.MedianTestForSnakeStats) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.002s

OK
```

Test Set 3: getMode() Function

This test set is used to develop a statistical function to compute the mode of a set of numbers in a list. The mode is the number in a list of numbers that appears the most often. TDD is used to incrementally build the getMode() function by writing a failing test and producing the minimal code to pass the test; this process is repeated. The first test tests output for a list of data with a single mode that appears as the first element. The second test tests output for a list of data with a single mode that appears in an arbitrary position. The third test tests output for a list of data with multiple modes that appear in arbitrary positions.

Test Set 3: Test 1

The purpose of this test is to determine if the getMode() function produces the correct output when the list of data contains a single mode that appears as the first element.

Create a failing test.

```
import unittest
import snakestats

class ModeTestForSnakeStats(unittest.TestCase):

    def test_single_mode_first_position(self):
        mode = snakestats.getMode([4,4,4,2,3,5,4,4,1])
        return self.assertEqual(mode,4)

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

Test failed because getMode() does not match any function signature in snakestats.py.

```
test_single_mode_first_position (__main__.ModeTestForSnakeStats) ... ERROR

======================================================================
ERROR: test_single_mode_first_position (__main__.ModeTestForSnakeStats)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-1-abffd992809f>", line 7, in test_single_mode_first_position
    mode = snakestats.getMode([4,4,42,3,5,4,4,1])
AttributeError: module 'snakestats' has no attribute 'getMode'

----------------------------------------------------------------------
Ran 1 test in 0.003s

FAILED (errors=1)
```

The minimal code needed to pass this test is to return the first element in the list.

```
1 def getMode(vals):
2     return vals[0]
```

Test successful.

```
test_single_mode_first_element (__main__.ModeTestForSnakeStats) ... ok

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

OK
```

Test Set 3: Test 2

The purpose of this test is to determine if the getMode() function produces the correct output when the list contains a single mode. The list's mode is no longer trivially positioned as the first element in the list, but instead in an arbitrary position.

Initial code.

```
1 def getMode(vals):
2     return vals[0]
```

Create a failing test.

```
import unittest
import snakestats

class ModeTestForSnakeStats(unittest.TestCase):

    def test_single_mode_first_position(self):
        mode = snakestats.getMode([4,4,4,2,3,5,4,4,1])
        return self.assertEqual(mode,4)

    def test_single_mode_arbitrary_position(self):
        mode = snakestats.getMode([2,4,4,4,3,5,4,4,1])
        return self.assertEqual(mode,4)

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

Test failed because getMedian() returns the median element assuming that the initial list is sorted.

```
test_single_mode_arbitrary_position (__main__.ModeTestForSnakeStats) ... FAIL
test_single_mode_first_position (__main__.ModeTestForSnakeStats) ... ok

======================================================================
FAIL: test_single_mode_arbitrary_position (__main__.ModeTestForSnakeStats)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-1-c84bbee065b1>", line 12, in test_single_mode_arbitrary_position
    return self.assertEqual(mode,4)
AssertionError: 2 != 4

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

FAILED (failures=1)
```

The minimal code needed to pass this test is to determine the frequency of each list element and store the output as a dictionary. Then, the element whose frequency corresponds to the maximum value in the frequency dictionary is acquired as a list. The first and only element in the list is returned.

```python
from collections import Counter

def getMode(vals):
    frequency = Counter(vals)
    findMode = dict(frequency)
    mode = [i for i, v in findMode.items() if v == max(list(frequency.values()))]
    return mode[0]
```

Test successful.

```
test_single_mode_arbitrary_position (__main__.ModeTestForSnakeStats) ... ok
test_single_mode_first_position (__main__.ModeTestForSnakeStats) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

Test Set 3: Test 3

The purpose of this test is to determine if the getMode() function produces the correct output when the list contains multiple modes in arbitrary positions.

Initial code.

```python
from collections import Counter

def getMode(vals):
    frequency = Counter(vals)
    findMode = dict(frequency)
    mode = [i for i, v in findMode.items() if v == max(list(frequency.values()))]
    return mode[0]
```

Create a failing test.

```python
import unittest
import snakestats

class ModeTestForSnakeStats(unittest.TestCase):

    def test_single_mode_first_position(self):
        mode = snakestats.getMode([4,4,4,2,3,5,4,4,1])
        return self.assertEqual(mode,4)

    def test_single_mode_arbitrary_position(self):
        mode = snakestats.getMode([2,4,4,4,3,5,4,4,1])
        return self.assertEqual(mode,4)

    def test_multiple_modes_arbitrary_positions(self):
        mode = snakestats.getMode([2,4,4,3,3,5,3,4,4,1,3])
        return self.assertEqual(mode,[4,3])

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

Test failed because getMode() returns a single mode. The expected output is a list of multiple modes with the same, maximum frequency of occurrence.

```
test_multiple_modes_arbitrary_positions (__main__.ModeTestForSnakeStats) ... FAIL
test_single_mode_arbitrary_position (__main__.ModeTestForSnakeStats) ... ok
test_single_mode_first_position (__main__.ModeTestForSnakeStats) ... ok

======================================================================
FAIL: test_multiple_modes_arbitrary_positions (__main__.ModeTestForSnakeStats)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-1-d2dd6c50457b>", line 16, in test_multiple_modes_arbitrary_positions
    return self.assertEqual(mode,[4,3])
AssertionError: 4 != [4, 3]

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

FAILED (failures=1)
```

The minimal code needed to pass this test involves determining the frequency of each element in the list. The frequency values are put in a dictionary and the elements that correspond to the maximum frequency of the dictionary converted to a list are put into a list. The ternary condition operator is needed in order for getMode() to accommodate single modes and multiple modes. If the length of the list is one, a single mode is outputted. If the length is greater than one, then multiple modes are outputted as a comma-delimited list.

```python
from collections import Counter

def getMode(vals):
    frequency = Counter(vals)
    findMode = dict(frequency)
    mode = [i for i, v in findMode.items() if v == max(list(frequency.values()))]
    return mode[0] if len(mode) == 1 else mode
```

Test successful.

```
test_multiple_modes_arbitrary_positions (__main__.ModeTestForSnakeStats) ... ok
test_single_mode_arbitrary_position (__main__.ModeTestForSnakeStats) ... ok
test_single_mode_first_position (__main__.ModeTestForSnakeStats) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

Part 3: Secure Programming

Secure Programming Problem and Solution: 1

The code extract is a GET request routing logic to query a database based on the request query item ID. It will select all products that match the given ID. Since an item ID tends to be the primary key in a database table, it is likely that a single item, or no items, are returned. This is a database query that was later revised in the 'Databases, Networks and the Web' module.

```
const db = require('./db');
app.get('/item', (req, res) => {
   db.query('SELECT * FROM items WHERE id = ' + req.query.id);
   .then((item) => {
   res.send(item);
```

```
  })
});
```

Problem

This code is vulnerable to an SQL injection attack vector discussed in *8.3.1 SQL injection* within the *8.3. Handle Metacharacters* section of *Secure Programming HOWTO* [1]. The corresponding Common Weakness Enumeration (CWE) weakness is *CWE-89: Improper Neutralization of Special Elements used in an SQL Command*. The SQL command uses an externally-influenced input from an upstream component that contains insufficiently neutralized special elements that modify the parsing or interpretation of the SQL command in the downstream component [1]. Incorrect filtering for the string literal escape character and weakly typed user input with unverified length boundaries enable injection of additional SQL statements and altering of querying logic via 'req.query.id' [2]. This may result in the disclosure of internal database structure, tampering with system data, and unauthorized administrative terminal and OS access. The lack of distinction between the control and data planes is due to untrusted user input being concatenated directly with the query string. The database query built using string concatenation is susceptible to security breaches, as user input can be interpreted as SQL as opposed to ordinary data.

Solution and Implementation

The *SQL Injection Prevention Cheat Sheet* developed by *The Open Web Application Security Project (OWASP)* recommends prepared statements with parameterized queries [3]. To distinguish between coding and data planes, the SQL code is defined first, and the parameters are passed later. The intent of the query is predetermined; if an attacker attempts to insert malicious SQL commands, the parameterized queries treat them as string literals. Another defense option is stored procedures, which function similarly to parameterized queries except that the SQL code is defined in the database, and then called via the application [4]. To disable code injection into dynamically generated queries, the same precautions of escaping and validation are required. The *OWASP Enterprise Security API (ESAPI)* is an open source web application security control library that contains the ESAPI database codec that provides SQL injection safety to dynamic queries. [5] User supplied parameters should be passed into the ESAPI.encoder().encodeForSQL(new OracleCodec(), queryparam) function as the query parameter. Another method of deterrence is to convert all input into hex codes corresponding to the ASCII/UTF-8 codes of the user data. This renders SQL that contains numeric digits and letters, and no special characters. This is effective, since user input channels are the main attack vector for an SQL injection. A final preventative measure is to enforce the principle of least privilege, which provides the users with the minimum permissions, or level of access, to perform their job functions [6].

Secure Programming Problem and Solution: 2

The code extract demonstrates the authorization of a user's inputted username and password fields and a subsequent redirection to a user-entered URL. This was an extension of the routing logic from the 'Databases, Networks and the Web' module.

```
const express = require('express');
const app = express();
const port = 8089;

app.post('/login', (req, res) => {
```

```
    const { username, password } = req.body;
    if (auth(username, password)) {
        res.redirect(req.query.redirect_url ? req.query.redirect_url : '/');
    } else {
        res.render('login', { error: 'Credentials could not be verified.' });
    }
});

app.listen(port, () => {
    console.log(`Listening on http://localhost:${port}`);
});
```

Problem

The web application uses user-provided input as a redirect to an external site without URL validation; this is discussed in *5.13.4. Validating Hypertext Links (URIs/URLs)* in *Secure Programming HOWTO* [7]. Open redirects enable malicious code injection via cross-site scripting, arbitrary outbound requests from a server (server-side request forgery (SSRF)), and illicitly obtained tokens via the referer HTTP header [8]. Based on *CWE-601: URL Redirection to Untrusted Site*, the URL modification can lead to the loss of user credential confidentiality, phishing scams, personally identifiable information (PII), and identity theft (via malware that conducts keylogging). The user is likely to trust the malicious site because the modified link points to the server name of the original trusted site with the malicious URL parameter from the POST request appended to it. Open redirects can be chained with other exploits to compromise a user's machine. This is critical for mobile users, since the majority of mobile devices display the domain, and obfuscate the parameters and redirections.

Solution and Implementation

The least flexible fix is to remove the parameter and use a fixed redirect. This disrupts the user experience by requiring manual navigation. A method that does not disrupt the flow of the application is to implement an "allow list", which specifies a series of endpoints that are valid; invalid redirect_url parameters are redirected to the home page [9]. A "block list" approach is not as restrictive as an explicitly indexed "allow list", which is why blacklisting is not preferred. This white-list approach can be used by providing valid hosts or using regex. The downside of white-list protection mechanisms is the vulnerability to overly permissive inputs, highlighted in *CWE-183: Permissive List of Allowed Inputs* [10]. The optimal method of reducing tampering is to have the user supply a token or ID which can be mapped to a target URL on the server via enforcement by conversion (i.e. using features of the ESAPI AccessReferenceMap) [11]. The main downside of this mapping is the possibility of an enumeration vulnerability, in which a user cycles through all IDs/tokens to find each redirect target [12]. If the web application contains many potential redirection endpoints, then it would be effective to implement a fixed domain with legitimate redirect pages appended to the end. The final approach is to force all redirection traffic to first go to a page that alerts users that they are leaving the main site, with a confirmation link and the destination address indicated.

Secure Programming Problem and Solution: 3

The code extract is part of the user-feedback portion of the web application developed in the 'Web Development' module. The section was embedded in the 'Contact Us' webpage within the site structure, and enabled users to

type input into a text input field and submit it The event listener which handles click events uses the .innerHTML property to update a dev to the value of the text inputted.

```
<section>
 <article id="text-input-article">
  <label for="text-input-field">User Feedback</label>
  <input id="text-input-field" type="text">
  <button id="submit-button">Submit</button>
  <div id=outputArea></div>
 </article>
</section>

textInput = document.getElementById('text-input-field');
submit = document.getElementById('submit-button');
output = document.getElementById('outputArea');

submit.addEventListener('click', () => { output.innerHTML = textInput.value; })
```

Problem

innerHTML injects HTML into an element via vanilla JavaScript. Since innerHTML renders markup as opposed to text, malicious code can be injected and executed in the site via a cross-site scripting (XSS) attack; this is discussed in *7.16. Prevent Cross-Site (XSS) Malicious Content* in *Secure Programming HOWTO* [13]. Since the user-controlled text input is not used to render static content in the DOM and is not controlled dynamic content, it becomes an attack vector. A fetch request can be used to provide access to the backend server as if it had originated from the host website. A malicious script targeting the user could be run during each page access: this could lead to account scraping from the browser. The relevant Common Weakness Enumeration (CWE) weakness is *CWE-79: Improper Neutralization of Input During Web Page Generation*, which describes how lack of neutralization of input can lead to malicious scripts and Dynamic HyperText Markup Language (DHTML) tags being executed that expose SSL-encrypted connections, create denial-of-service attacks through infinite window creation, and buffer overflows.

Solution and Implementation

In cases in which text can be rendered in place of HTML, .innerText or .textContent can be used. The textContent property is compatible with all modern browsers, and IE9 and above [14]. This renders the input HTML to the DOM as plain text by escaping malicious content, which removes script execution and DOM-based XSS [15]. If parsing HTML is required, then using an HTML sanitizer such as DOMPurify is imperative. DOMPurify is an XSS filter that scales with the speed of the browser. This provides a more detailed look into the sanitization process, as DOMPurify.removed indicates the elements and attributes that were removed [16]. The setHTML() method belonging to the Element interface can also be used to parse and sanitize unwanted XSS attributes. Moreover, Node.js has built in encodeURI and encodeURIComponent global functions that are designed to escape XSS vulnerabilities for full URLs and query strings, respectively [17]. Using frameworks with built in safe methods or advanced packages such as xss-filters from the npm package in Express applications can remove markup from the code. It is critical that these client side security considerations are implemented on the server side in order to avoid *CWE-602: Client-Side Enforcement of Server-Side Security* [18]. An attack surface reduction method can be

implemented by setting the session cookie to be HttpOnly. This attribute removes the session cookie's accessibility to client-side scripts that use document.cookie. Moreover, it is essential to specify the output encoding to be handled by the downstream component: common encodings such as UTF-8 and ISO-8859-1 should be used to prevent encoding inconsistency, which can result in injection attacks [19]. For example, control characters and special elements should be translated into safe form that cannot be destructive to a target interpreter via output encoding.

References

[1] https://cwe.mitre.org/data/definitions/89.html
[2] https://success.outsystems.com/Documentation/11/Reference/Errors_and_Warnings/Warnings
[3] https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
[4] https://www.ecanarys.com/Blogs/ArticleID/112/SQL-injection-attack-and-prevention-using-stored-procedure
[5] https://owasp.org/www-project-enterprise-security-api/
[6] https://www.cyberark.com/what-is/least-privilege/
[7] https://dwheeler.com/secure-programs/Secure-Programs-HOWTO/filter-html.html
[8] https://www.invicti.com/blog/web-security/open-redirect-vulnerabilities-invicti-pauls-security-weekly/
[9] https://www.techtarget.com/whatis/definition/whitelist
[10] https://cwe.mitre.org/data/definitions/183.html
[11] https://cwe.mitre.org/data/definitions/829.html
[12] https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing
[13] https://dwheeler.com/secure-programs/Secure-Programs-HOWTO/cross-site-malicious-content.html
[14] https://www.lambdatest.com/web-technologies/textcontent-support-on-ie-9
[15]https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/innerText
[16] https://www.npmjs.com/package/dompurify
[17] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encodeURIComponent
[18] https://cwe.mitre.org/data/definitions/602.html
[19] https://www.techfolks.net/utf-8-vs-iso-8859-1/