TDD FileLoader v1.3

Unit test modified so it is build pipeline friendly and does not break the tenants of good Unit Testing

Unit Test

```
import unittest
from app.file_loader import FileLoader

class TestFileLoader(unittest.TestCase):

    def test_load_all_of_file_using_inbuilt_files_type_as_lambda(self):
        # Arrange
        file_to_load = "sample.txt"
        cut = FileLoader(file_to_load)
        expected_bytes_read = 12

        # Calculate expected number of characters
        with open(file_to_load, encoding="utf-8") as f:
            expected_bytes_read = sum(len(line) for line in f.readlines())

        # Act
        bytes_read = cut.load_file_with_func(lambda fname: open(fname,
                                    encoding="utf-8").readlines())

        # Assert
        self.assertEqual(expected_bytes_read, bytes_read)

    def test_load_all_of_file_via_stub(self):
        """ Use a hardcoded stub to simulate reading two lines of text
            Benefit - no dependencyon actual files or filesystem
                    - portable test
                    - FileLoader is more flexible and decoupled allowing
                      file loading mechanism to be injected
        """
        # arrange
        file_to_load = ""
        cut = FileLoader(file_to_load)
        expected_bytes_read = 10

        # act
        bytes_read = cut.load_file_with_func(lambda fname: ["Hello", "world"])

        # assert
        self.assertEqual(expected_bytes_read, bytes_read)
```

The CUT FileLoader

```
"""
FileLoader Module

This module defines the FileLoader class which is responsible for reading a text file
and calculating the total size (in characters) of its contents. It also supports
dependency injection through the `load_file_with_func` method, allowing testability
without relying on actual file I/O operations.

This is useful in unit testing scenarios where file system access should be avoided.
"""

class FileLoader:
    def __init__(self, file_to_load):
        self.file_to_load = file_to_load
        self.lines = []

    def load_file(self, fname):
        """
        Loads a file from disk and reads its contents line by line.
        Falls back to an empty list if the file cannot be read.
        """
        try:
            with open(fname, encoding='utf-8') as f:
                self.lines = f.readlines()
        except IOError:
            self.lines = []
        return self._calculate_file_size()

    def get_lines(self):
        """Returns the list of lines read from the file."""
        return self.lines

    def load_file_with_func(self, func):
        """
        Accepts a file loading function to inject lines, used primarily for testing.

        This avoids direct I/O operations and makes the method more testable by passing
        a mock or simulated version of file loading logic.
        """
        self.lines = func(self.file_to_load)
        return self._calculate_file_size()

    def _calculate_file_size(self):
        return sum(len(line) for line in self.lines)
```