Harrison_Beard_ProbSet1

June 26, 2018

1 Intro to NumPy

Note: If any of the lines of code are too long to fit on the page, then please reference the .py or the .tex file with this same title, in this directory.

Problem 1

```
In [3]: import numpy as np

        ### PART 1: INTRO TO NUMPY

        # Problem 1
        def problem1_1():
            A = np.array([[3, -1,  4],
                          [1,  5, -9]])
            B = np.array([[2,  6, -5,  3],
                          [5, -8,  9,  7],
                          [9, -3, -2, -3]])
            return np.dot(A, B)
```

Problem 2

```
In [4]: # Problem 2
        def problem1_2():
            A = np.array([[3,  1,  4],
                          [1,  5,  9],
                          [-5, 3,  1]])
            ans = -1 * (np.dot(A, np.dot(A, A))) + 9 * np.dot(A, A) - 15 * A
            return ans
```

Problem 3

```
In [5]: # Problem 3
        def problem1_3():
            ones = np.ones((7,7))
            A = np.triu(ones)
            fives = np.full((7,7), 5) - 5 * np.eye(7)
            negOnes = -1 * ones
```

```
    B = np.triu(fives) + np.tril(negOnes)
    ABA = np.dot(A, np.dot(B, A))
    ABA = ABA.astype(np.int64)
    return ABA
```

Problem 4

```
In [6]: # Problem 4
        def problem1_4(array):
            arrayCopy = array
            mask = arrayCopy < 0
            arrayCopy[mask] = 0
            return arrayCopy
```

Problem 5

```
In [7]: # Problem 5
        def problem1_5():
            A = np.array([[0, 2 ,4],
                          [1 ,3 ,5]])
            B = np.full((3,3), 3)
            B = np.tril(B)
            C = -2 * np.eye(3)
            row1 = np.hstack((np.zeros((3,3)), A.T, np.eye(3)))
            row2 = np.hstack((A, np.zeros((2,2)), np.zeros((2,3))))
            row3 = np.hstack((B, np.zeros((3,2)), C))
            output = np.vstack((row1, row2, row3))
            return output
```

Problem 6

Transposing worked better than reshaping here.

```
In [8]: # Problem 6
        def problem1_6(matrix):
            return matrix / matrix.sum(axis=1).reshape((2,1))
```

Problem 7

This is a natural extension to the provided example, for all the four directions.

```
In [26]: # Problem 7
         def problem1_7():
             grid = np.load("grid.npy")
             hMax  = np.max(grid[:,:-3]   * grid[:,1:-2]    * grid[:,2:-1]    * grid[:,3:])
             vMax  = np.max(grid[:-3,:]   * grid[1:-2,:]    * grid[2:-1,:]    * grid[3:,:])
             dMax  = np.max(grid[:-3,:-3] * grid[1:-2,1:-2] * grid[2:-1,2:-1] * grid[3:,3:])
             idMax = np.max(grid[:-3,3:] * grid[1:-2,2:-1] * grid[2:-1,1:-2] * grid[3:,:-3])
             gMax = max(hMax, vMax, dMax, idMax)
```

```
        print("horizontal max: " + str(hMax))
        print("vertical max: " + str(vMax))
        print("diagonal (L to R, T to B) max: " + str(dMax))
        print("inverse-diagonal (R to L, T to B) max: " + str(idMax))
        print("global max: " + str(gMax))
        return gMax
```

2 Standard Library

Problem 1

Returning in one line as a tuple, we have:

```
In [11]: ### PART 2: STANDARD LIBRARY

         # Problem 1
         def problem2_1(L):
             return (min(L), max(L), sum(L)/len(L)) # (min, max, avg)
```

Problem 2

```
In [58]: # Problem 2
         def problem2_2():
             int_1 = 5
             int_2 = int_1
             int_2 += 1
             print("Integers are mutable: " + str(int_1 == int_2))

             str_1 = "hello"
             str_2 = str_1
             str_2 += " world"
             print("Strings are mutable: " + str(str_1 == str_2))

             list_1 = [1,2,3]
             list_2 = list_1
             list_2 += [4]
             print("Lists are mutable: " + str(list_1 == list_2))

             tuple_1 = (1,2,3)
             tuple_2 = tuple_1
             tuple_2 += (4,)
             print("Tuples are mutable: " + str(tuple_1 == tuple_2))

             set_1 = {1,2,3}
             set_2 = set_1
             set_2.add(4)
             print("Sets are mutable: " + str(set_1 == set_2))
```

The result:

```
In [59]: problem2_2()
```

```
Integers are mutable: False
Strings are mutable: False
Lists are mutable: True
Tuples are mutable: False
Sets are mutable: True
```

Problem 3

```
In [69]: # Calculator.py

         from math import sqrt

         def add(a,b):
             return a+b

         def mult(a,b):
             return a*b

         if __name__=="__main__":
             pass # in case I wanted to execute anything from the command line or interpreter, I
         else:
             pass # if it were imported, I would put it the execution code here.
```

Now, we implement $\sqrt{a^2+b^2}$:

```
In [70]: # Problem 3
         import calculator as c
         def problem2_3(a,b):
             return c.sqrt(c.add(c.mult(a,a),c.mult(b,b)))
```

Putting this to the test, we see that $\sqrt{1^2+1^2}$ does ineed equal $\sqrt{2}$:

```
In [72]: problem2_3(1,1)
```

```
Out[72]: 1.4142135623730951
```

Problem 4

```
In [39]: # Problem 4
         from itertools import combinations
         def problem2_4(A):
             powerset = []
             for i in range(len(A)+1):
                 subset = combinations(A,i)
                 powerset += subset
             for j in range(len(powerset)):
                 powerset[j] = set(powerset[j])
             return powerset
```

Unfortunately, sets can't contain interables, so we must construct a *list* of sets. (Should we call it a power-list instead?)

Problem 5

I decided to challenge myself and implement my own custom-made functions, rather than importing the provided box module. See below.

```
In [ ]: import box
        import time
        import sys
        import random
        from itertools import combinations

        def power(A):
            """
            but no emptyset included, since that wouldn't help in this context.
            """
            powerset = []
            for i in range(1,len(A)+1):
                subset = combinations(A,i)
                powerset += subset
            for j in range(len(powerset)):
                powerset[j] = set(powerset[j])
            return powerset

        def prompt(numbers, startTime, timeLimit, roll):
            """
            Continually prompts the user for numbers to eliminate
            """
            secondsRemaining = round(startTime + timeLimit - time.time(),2)
            if secondsRemaining <= 0:
                return "out of time"
            print("Seconds left: " + str(secondsRemaining))
            entry = input("Numbers to eliminate: ")


            numsToEliminate = []
            shouldBeSpace = False

            numbersAsStrings = []
            for j in numbers:
                numbersAsStrings.append(str(j))

            numsToEliminateAsStrings = []

            for i in entry:
                if (i != " " and shouldBeSpace) or (i not in numbersAsStrings and not shouldBeSp
```

```
            print("Invalid input")
            return "invalid"
        elif not shouldBeSpace:
            for k in numsToEliminate:
                numsToEliminateAsStrings.append(str(k))
            if i in numsToEliminateAsStrings:
                print("Invalid input")
                return "invalid"
            else:
                numsToEliminate.append(int(i))

        shouldBeSpace = not shouldBeSpace

    if sum(numsToEliminate) != roll:
        print("Invalid input")
        return "invalid"

    return numsToEliminate


def main():
    """
    Runs the game. Needs two arguments: a player name and a time limit.
    """
    playerName = sys.argv[1]
    timeLimit = float(sys.argv[2])

    numbers = set(range(1,10))

    numsToEliminate = []

    roll = 0

    elapsedTime = 0.0

    startTime = time.time()

    lose = False

    while (sum(numbers) > 0) and (not lose):
        elapsedTime = time.time() - startTime

        possibleChoices = power(numbers)
        for i in range(len(possibleChoices)):
            possibleChoices[i] = sum(possibleChoices[i])

        if elapsedTime > timeLimit:
            print("Game over!")
```

```
            lose = True
            break

        if sum(numbers) > 6:
            roll = random.choice(list(range(1,7))) + random.choice(list(range(1,7)))
        else:
            roll = random.choice(list(range(1,7)))

        print("Numbers left: " + str(numbers))
        print("Roll: " + str(roll))

        if roll not in possibleChoices:
            print("Game over!\n")
            lose = True
            break

        invalid = True
        while invalid:
            numsToEliminate = prompt(numbers, startTime, timeLimit, roll)
            print("")
            if numsToEliminate == "invalid":
                invalid = True
            elif numsToEliminate == "out of time":
                print("Game over!\n")
                lose = True
                invalid = False
            else:
                invalid = False

        if not lose:
            for i in numsToEliminate:
                numbers.remove(i)


    elapsedTime = time.time() - startTime

    if sum(numbers) == 0:
        print("Score for player " + playerName + ": " + str(sum(numbers)) + " points")
        print("Time played: " + str(round(elapsedTime,2)) + " seconds")
        print("Congratulations!! You shut the box! :)")

    if lose:
        print("Score for player " + playerName + ": " + str(sum(numbers)) + " points")
        print("Time played: " + str(round(elapsedTime,2)) + " seconds")
        print("Better luck next time! >:)")


if __name__ == "__main__" and len(sys.argv) == 3:
```

```
        main() # only run main() if called from command line or interpreter directly

    else:
        print("Exactly two extra command line argument is required")
        print("System Arguments:", sys.argv2)
```

Although this script doesn't seem to work from the Jupyter notebook, it works perfectly when i run it through my shell command line. It's saved as `standard_library.py` in this directory.

3 Unit Testing

Problem 1

Installing `pytest` and some related packages, using the ! flag to run BASH:

```
In [316]: !pip install msgpack
```

```
Requirement already satisfied: msgpack in /anaconda3/lib/python3.6/site-packages (0.5.6)
```

```
In [329]: !pip install -U pytest
```

```
The following command must be run outside of the IPython shell:

    $ pip install -U pytest

The Python package manager (pip) can only be used from outside of IPython.
Please reissue the `pip` command in a separate terminal or command prompt.

See the Python documentation for more information on how to install packages:

    https://docs.python.org/3/installing/
```

Original program:

```
In [ ]: # # smallest_factor.py

        # def smallest_factor(n):
        #     """Return smallest prime factor of pos int n"""
        #     if n==1: return 1
        #     for i in range(2,int(n**.5)):
        #         if n % i == 0 : return i
        #     return n
```

```
In [91]: !py.test
```

```
============================= test session starts ==============================
platform darwin -- Python 3.6.5, pytest-3.6.2, py-1.5.3, pluggy-0.6.0
rootdir: /Users/hbeard/Documents/GitHub/BootCamp2018/ProbSets/Comp/ProbSet1, inifile:
plugins: remotedata-0.2.1, openfiles-0.3.0, doctestplus-0.1.3, cov-2.5.1, arraydiff-0.2
collected 34 items                                                             test_fraction.py
test_month_length.py ...                                                 [ 44%]
test_operate.py ...                                                      [ 64%]
test_setgame.py ...                                                      [ 82%]
test_smallest_factor.py ...                                              [100%]

========================== 34 passed in 0.13 seconds ===========================
```

New, corrected program:

```
In [ ]: # # smallest_factor.py

        # def smallest_factor(n):
        #     """Return smallest prime factor of pos int n"""
        #     if n==1: return 1
        #     for i in range(2,int(n**.5)+1):
        #         if n % i == 0 : return i
        #     return n
```

What I did here was add one to the upper bound on the `range()` function, since only having `int(n**.5)` would mean that $\sqrt{n}$ would be set to $\lfloor\sqrt{n}\rfloor$, which means that Python would index only up to $\lfloor\sqrt{n}\rfloor - 1$; thus, a $+1$ was neeeded to justify for that error.

```
In [344]: !py.test

============================= test session starts ==============================
platform darwin -- Python 3.6.5, pytest-3.6.2, py-1.5.3, pluggy-0.6.0
rootdir: /Users/hbeard/Documents/GitHub/BootCamp2018/ProbSets/Comp/ProbSet1, inifile:
plugins: remotedata-0.2.1, openfiles-0.3.0, doctestplus-0.1.3, arraydiff-0.2
collected 6 items

test_smallest_factor.py ...                                              [100%]

========================== 6 passed in 0.02 seconds ============================
```

Luckily, my correction works!

Problem 2

Testing, we have

```
In [351]: !py.test --cov
```

```
============================= test session starts ==============================
platform darwin -- Python 3.6.5, pytest-3.6.2, py-1.5.3, pluggy-0.6.0
rootdir: /Users/hbeard/Documents/GitHub/BootCamp2018/ProbSets/Comp/ProbSet1, inifile:
plugins: remotedata-0.2.1, openfiles-0.3.0, doctestplus-0.1.3, cov-2.5.1, arraydiff-0.2
collected 6 items                                                              test_smallest_fac

---------- coverage: platform darwin, python 3.6.5-final-0 -----------
Name                        Stmts   Miss  Cover
---------------------------------------------
smallest_factor.py              5      0   100%
test_smallest_factor.py        13      0   100%
---------------------------------------------
TOTAL                          18      0   100%


========================== 6 passed in 0.05 seconds ===========================
```

I have 100% coverage, so I need not write additional test cases.

```
In [ ]: # test_month_length.py

        import month_length as ml

        def test_30DayNonLeap():
            assert ml.month_length("April") == 30, "failed on 30-day month "+\
                                                   " on a non-leap year"
        def test_31DayNonLeap():
            assert ml.month_length("January") == 31, "failed on 31-day month "+\
                                                     " on a non-leap year"

        def test_FebNonLeap():
            assert ml.month_length("February") == 28, "failed on Februrary "+\
                                                      " on a non-leap year"

        def test_30DayLeap():
            assert ml.month_length("April",leap_year=True) == 30, "failed on "+\
                                                    "30-day month on a leap year"

        def test_31DayLeap():
            assert ml.month_length("January",leap_year=True) == 31, "failed on "+\
                                                   "31-day month on a leap year"

        def test_FebLeap():
            assert ml.month_length("February",leap_year=True) == 29, "failed on "+\
                                                     "Februrary on a leap year"

        def test_notAMonth():
            assert ml.month_length("X") == None, "failed on something that is not a month"
```

```
In [372]: !py.test --cov

============================= test session starts ==============================
platform darwin -- Python 3.6.5, pytest-3.6.2, py-1.5.3, pluggy-0.6.0
rootdir: /Users/hbeard/Documents/GitHub/BootCamp2018/ProbSets/Comp/ProbSet1, inifile:
plugins: remotedata-0.2.1, openfiles-0.3.0, doctestplus-0.1.3, cov-2.5.1, arraydiff-0.2
collected 13 items                                                             test_month_length
test_smallest_factor.py ...                                                  [100%]

---------- coverage: platform darwin, python 3.6.5-final-0 -----------
Name                       Stmts   Miss  Cover
--------------------------------------------
month_length.py               10      0   100%
smallest_factor.py             5      0   100%
test_month_length.py          15      0   100%
test_smallest_factor.py       13      0   100%
--------------------------------------------
TOTAL                         43      0   100%


========================== 13 passed in 0.06 seconds ===========================
```

Looks like it worked!

Problem 3

```
In [ ]: # test_operate.py

        import operate as op
        import pytest

        def test_typeError():
            with pytest.raises(TypeError) as excinfo:
                op.operate(1,2,3)
            assert excinfo.value.args[0] == "oper must be a string"

        def test_add():
            assert op.operate(8,4,"+") == 12, "failed on '+' operation"

        def test_sub():
            assert op.operate(8,4,"-") == 4, "failed on '-' operation"

        def test_mul():
            assert op.operate(8,4,"*") == 32, "failed on '*' operation"

        def test_truediv():
            assert op.operate(8,4,"/") == 2, "failed on '/' operation"
```

```
        def test_zeroDivisionError():
            with pytest.raises(ZeroDivisionError) as excinfo:
                op.operate(1,0,"/")
            assert excinfo.value.args[0] == "division by zero is undefined"

        def test_valueError():
            with pytest.raises(ValueError) as excinfo:
                op.operate(1,2,"hello")
            assert excinfo.value.args[0] == "oper must be one of '+', '/', '-', or '*'"
```

```
In [380]: !py.test --cov

============================= test session starts ==============================
platform darwin -- Python 3.6.5, pytest-3.6.2, py-1.5.3, pluggy-0.6.0
rootdir: /Users/hbeard/Documents/GitHub/BootCamp2018/ProbSets/Comp/ProbSet1, inifile:
plugins: remotedata-0.2.1, openfiles-0.3.0, doctestplus-0.1.3, cov-2.5.1, arraydiff-0.2
collected 20 items                                                              test_month_length
test_operate.py ...                                                    [ 70%]
test_smallest_factor.py ...                                            [100%]

---------- coverage: platform darwin, python 3.6.5-final-0 -----------
Name                       Stmts   Miss  Cover
--------------------------------------------
month_length.py               10      0   100%
operate.py                    14      0   100%
smallest_factor.py             5      0   100%
test_month_length.py          15      0   100%
test_operate.py               22      0   100%
test_smallest_factor.py       13      0   100%
--------------------------------------------
TOTAL                         79      0   100%


========================== 20 passed in 0.08 seconds ===========================


In [379]: !py.test --cov-report html --cov

============================= test session starts ==============================
platform darwin -- Python 3.6.5, pytest-3.6.2, py-1.5.3, pluggy-0.6.0
rootdir: /Users/hbeard/Documents/GitHub/BootCamp2018/ProbSets/Comp/ProbSet1, inifile:
plugins: remotedata-0.2.1, openfiles-0.3.0, doctestplus-0.1.3, cov-2.5.1, arraydiff-0.2
collected 20 items                                                              test_month_length
test_operate.py ...                                                    [ 70%]
test_smallest_factor.py ...                                            [100%]

---------- coverage: platform darwin, python 3.6.5-final-0 -----------
Coverage HTML written to dir htmlcov
```

```
========================== 20 passed in 0.12 seconds ==========================
```

It looks like it all worked!

Problem 4

First of all, the two errors were in `__add__()` and `__sub__()`.

```
In [ ]: # fraction.py

        class Fraction(object):
            """Reduced fraction class with integer numerator and denominator."""
            def __init__(self, numerator, denominator):
                if denominator == 0:
                    raise ZeroDivisionError("denominator cannot be zero")
                elif type(numerator) is not int or type(denominator) is not int:
                    raise TypeError("numerator and denominator must be integers")
                def gcd(a,b):
                    while b != 0:
                        a, b = b, a % b
                    return a
                common_factor = gcd(numerator, denominator)
                self.numer = numerator // common_factor
                self.denom = denominator // common_factor
            def __str__(self):
                if self.denom != 1:
                    return "{} / {}".format(self.numer, self.denom)
                else:
                    return str(self.numer)
            def __float__(self):
                return self.numer / self.denom
            def __eq__(self, other):
                if type(other) is Fraction:
                    return self.numer==other.numer and self.denom==other.denom
                else:
                    return float(self) == other
            def __add__(self, other):
                return Fraction(self.numer*other.denom + self.denom*other.numer, self.denom*othe
            def __sub__(self, other):
                return Fraction(self.numer*other.denom - self.denom*other.numer,
            self.denom*other.denom)
            def __mul__(self, other):
                return Fraction(self.numer*other.numer, self.denom*other.denom)
            def __truediv__(self, other):
                if self.denom*other.numer == 0:
```

```
                raise ZeroDivisionError("cannot divide by zero")
            return Fraction(self.numer*other.denom, self.denom*other.numer)
```

In [ ]:
```
# test_fraction.py

import fraction as fr

import pytest

@pytest.fixture
def set_up_fractions():
    """
    Sets up sample fractions for use in the tests.
    """
    frac_1_3 = fr.Fraction(1, 3)
    frac_1_2 = fr.Fraction(1, 2)
    frac_n2_3 = fr.Fraction(-2, 3)
    return frac_1_3, frac_1_2, frac_n2_3

def test_fraction_init(set_up_fractions):
    """
    Tests the initialization of Fraction objects.
    """
    frac_1_3, frac_1_2, frac_n2_3 = set_up_fractions
    assert frac_1_3.numer == 1
    assert frac_1_2.denom == 2
    assert frac_n2_3.numer == -2
    frac = fr.Fraction(30, 42) # 30/42 reduces to 5/7.
    assert frac.numer == 5
    assert frac.denom == 7
    with pytest.raises(ZeroDivisionError) as excinfo:
        fr.Fraction(1,0)
    assert excinfo.value.args[0] == "denominator cannot be zero"
    with pytest.raises(TypeError) as excinfo:
        fr.Fraction(1,"2")
    assert excinfo.value.args[0] == "numerator and denominator must be integers"

def test_fraction_str(set_up_fractions):
    """
    Tests the __str__() magic method
    """
    frac_1_3, frac_1_2, frac_n2_3 = set_up_fractions
    assert str(frac_1_3) == "1 / 3"
    assert str(frac_1_2) == "1 / 2"
    assert str(frac_n2_3) == "-2 / 3"
    assert str(fr.Fraction(2,1)) == "2"

def test_fraction_float(set_up_fractions):
```

```
    """
    Tests the __float__() magic method
    """
    frac_1_3, frac_1_2, frac_n2_3 = set_up_fractions
    assert float(frac_1_3) == 1 / 3.
    assert float(frac_1_2) == .5
    assert float(frac_n2_3) == -2 / 3.
    assert float(frac_1_3) == frac_1_3.numer / frac_1_3.denom

def test_fraction_eq(set_up_fractions):
    """
    Tests the __eq__() magic method
    """
    frac_1_3, frac_1_2, frac_n2_3 = set_up_fractions
    assert frac_1_2 == fr.Fraction(1, 2)
    assert frac_1_3 == fr.Fraction(2, 6)
    assert frac_n2_3 == fr.Fraction(8, -12)
    other = fr.Fraction(1, 2)
    assert (frac_1_2 == other) == (frac_1_2.numer == other.numer) and \
           (frac_1_2.denom == other.denom)
    assert (frac_1_2 == .5) == (float(frac_1_2) == .5)

def test_fraction_add(set_up_fractions):
    """
    Tests the __add__() magic method
    """
    frac_1_3, frac_1_2, frac_n2_3 = set_up_fractions
    assert frac_1_2 + frac_1_3 == fr.Fraction(frac_1_2.numer * frac_1_3.denom + \
                                              frac_1_2.denom * frac_1_3.numer, \
                                              frac_1_2.denom * frac_1_3.denom)
    assert frac_n2_3 + frac_1_2 == fr.Fraction(frac_n2_3.numer * frac_1_2.denom + \
                                              frac_n2_3.denom * frac_1_2.numer, \
                                              frac_n2_3.denom * frac_1_2.denom)

def test_fraction_sub(set_up_fractions):
    """
    Tests the __sub__() magic method
    """
    frac_1_3, frac_1_2, frac_n2_3 = set_up_fractions
    assert frac_1_2 - frac_1_3 == fr.Fraction(frac_1_2.numer * frac_1_3.denom - \
                                              frac_1_2.denom * frac_1_3.numer, \
                                              frac_1_2.denom * frac_1_3.denom)
    assert frac_n2_3 - frac_1_2 == fr.Fraction(frac_n2_3.numer * frac_1_2.denom - \
                                              frac_n2_3.denom * frac_1_2.numer, \
                                              frac_n2_3.denom * frac_1_2.denom)

def test_fraction_mul(set_up_fractions):
    """
```

```
            Tests the __mul__() magic method
            """
            frac_1_3, frac_1_2, frac_n2_3 = set_up_fractions
            assert frac_1_2 * frac_1_3 == fr.Fraction(frac_1_2.numer * frac_1_3.numer, \
                                                      frac_1_2.denom * frac_1_3.denom)
            assert frac_n2_3 * frac_1_2 == fr.Fraction(frac_n2_3.numer * frac_1_2.numer, \
                                                      frac_n2_3.denom * frac_1_2.denom)

        def test_fraction_truediv(set_up_fractions):
            """
            Tests the __truediv__() magic method
            """
            frac_1_3, frac_1_2, frac_n2_3 = set_up_fractions
            assert frac_1_2 / frac_1_3 == fr.Fraction(frac_1_2.numer * frac_1_3.denom, \
                                                      frac_1_2.denom * frac_1_3.numer)
            assert frac_n2_3 / frac_1_2 == fr.Fraction(frac_n2_3.numer * frac_1_2.denom, \
                                                      frac_n2_3.denom * frac_1_2.numer)
            with pytest.raises(ZeroDivisionError) as excinfo:
                frac_1_3 / fr.Fraction(0,1)
            assert excinfo.value.args[0] == "cannot divide by zero"
```

```
In [398]: !py.test --cov
```

```
============================= test session starts ==============================
platform darwin -- Python 3.6.5, pytest-3.6.2, py-1.5.3, pluggy-0.6.0
rootdir: /Users/hbeard/Documents/GitHub/BootCamp2018/ProbSets/Comp/ProbSet1, inifile:
plugins: remotedata-0.2.1, openfiles-0.3.0, doctestplus-0.1.3, cov-2.5.1, arraydiff-0.2
collected 28 items                                                             test_fraction.py
test_month_length.py ...                                                  [ 53%]
test_operate.py ...                                                       [ 78%]
test_smallest_factor.py ...                                               [100%]

---------- coverage: platform darwin, python 3.6.5-final-0 -----------
Name                      Stmts   Miss  Cover
---------------------------------------------
fraction.py                  33      0   100%
month_length.py              10      0   100%
operate.py                   14      0   100%
smallest_factor.py            5      0   100%
test_fraction.py             60      0   100%
test_month_length.py         15      0   100%
test_operate.py              22      0   100%
test_smallest_factor.py      13      0   100%
---------------------------------------------
TOTAL                       172      0   100%


========================== 28 passed in 0.27 seconds ===========================
```

```
In [399]: !py.test --cov-report html --cov

============================= test session starts ==============================
platform darwin -- Python 3.6.5, pytest-3.6.2, py-1.5.3, pluggy-0.6.0
rootdir: /Users/hbeard/Documents/GitHub/BootCamp2018/ProbSets/Comp/ProbSet1, inifile:
plugins: remotedata-0.2.1, openfiles-0.3.0, doctestplus-0.1.3, cov-2.5.1, arraydiff-0.2
collected 28 items                                                             test_fraction.py
test_month_length.py ...                                                  [ 53%]
test_operate.py ...                                                       [ 78%]
test_smallest_factor.py ...                                               [100%]

---------- coverage: platform darwin, python 3.6.5-final-0 -----------
Coverage HTML written to dir htmlcov


========================== 28 passed in 0.37 seconds ===========================
```

Looks like the tests were effective!

Problem 5

Here is the unit testing file:

```
In [ ]: # test_setgame.py

        import setgame as s

        from itertools import combinations

        import pytest

        @pytest.fixture
        def set_up_cards():
            """
            Sets up exemplary hands that could be helpful for unit testing.
            """
            cards_1 = ["0000","0001","0002","0010",
                       "0011","0012","0020","0021",
                       "0022","0100","0101","0102"]
            cards_2 = ["1000","1001","1002","1010",
                       "1011","1012","1020","1021",
                       "1022","1100","1101","1102"]
            cards_3 = ["2000","2001","2002","2010",
                       "2011","2012","2020","2021",
                       "2022","2100","2101","2102"]
            return cards_1, cards_2, cards_3
```

```
def test_count_sets_ve1():
    """
    Tests for the first kind of ValueError. - using fewer than 12 cards.
    """
    with pytest.raises(ValueError) as excinfo:
        s.count_sets(["0000"])
    assert excinfo.value.args[0] == "Please enter a list of exactly 12 unique cards, eac

def test_count_sets_ve2(set_up_cards):
    """
    Tests for the second kind of ValueError. - using non-unique cards.
    """
    cards_1, cards_2, cards_3 = set_up_cards
    with pytest.raises(ValueError) as excinfo:
        s.count_sets(["0000","0000","0002","0010",
                "0011","0012","0020","0021",
                "0022","0100","0101","0102"])
    assert excinfo.value.args[0] == "Please enter a list of exactly 12 unique cards, eac

def test_count_sets_ve3(set_up_cards):
    """
    Tests for the third kind of ValueError. - using strings that aren't four chars long.
    """
    cards_1, cards_2, cards_3 = set_up_cards
    with pytest.raises(ValueError) as excinfo:
        s.count_sets(["000","001","002","000",
                "001","12","000","001",
                "022","000","0101","00"])
    assert excinfo.value.args[0] == "Please enter a list of exactly 12 unique cards, eac

def test_count_sets_ve4(set_up_cards):
    """
    Tests for the fourth kind of ValueError. - using a digit that's not a 0, 1, or 2.
    """
    cards_1, cards_2, cards_3 = set_up_cards
    with pytest.raises(ValueError) as excinfo:
        s.count_sets(["5000","0001","0002","0010",
                "0011","0012","0020","0021",
                "0022","0100","0101","0102"])
    assert excinfo.value.args[0] == "Please enter a list of exactly 12 unique cards, eac


def test_count_sets_counting(set_up_cards):
    """
    Tests the count_sets function
    """
    cards_1, cards_2, cards_3 = set_up_cards
    sets_1 = 0
```

```
    sets_2 = 0
    sets_3 = 0
    for i in combinations(cards_1,3):
        if s.is_set(i[0],i[1],i[2]):
            sets_1+=1
    for i in combinations(cards_2,3):
        if s.is_set(i[0],i[1],i[2]):
            sets_2+=1
    for i in combinations(cards_3,3):
        if s.is_set(i[0],i[1],i[2]):
            sets_3+=1

    assert s.count_sets(cards_1) == sets_1, "failed at counting sets"
    assert s.count_sets(cards_2) == sets_2, "failed at counting sets"
    assert s.count_sets(cards_3) == sets_3, "failed at counting sets"


def test_is_set():
    """
    Tests the is_set function
    """
    abc_1 = ["1001","1001","2002"] # not a set
    abc_2 = ["1001","1001","2001"] # not a set
    abc_3 = ["0000","0001","1001"] # not a set
    isset1, isset2, isset3 = True, True, True
    for i in range(4):

        print("1")
        print(set([abc_1[0][i], abc_1[1][i], abc_1[2][i]]))
        if len(set([abc_1[0][i], abc_1[1][i], abc_1[2][i]])) not in [1,3]:
            isset1 = False

        print("2")
        print(set([abc_2[0][i], abc_2[1][i], abc_2[2][i]]))
        if len(set([abc_2[0][i], abc_2[1][i], abc_2[2][i]])) not in [1,3]:
            isset2 = False

        print("3")
        print(set([abc_3[0][i], abc_3[1][i], abc_3[2][i]]))
        if len(set([abc_3[0][i], abc_3[1][i], abc_3[2][i]])) not in [1,3]:
            isset3 = False
    assert s.is_set(abc_1[0],abc_1[1],abc_1[2]) == isset1, "failed at determining if a s
    assert s.is_set(abc_2[0],abc_2[1],abc_2[2]) == isset2, "failed at determining if a s
    assert s.is_set(abc_3[0],abc_3[1],abc_3[2]) == isset3, "failed at determining if a s
```

Problem 6

And here is the implementation:

```
In [ ]: # setgame.py

        from itertools import combinations

        def count_sets(cards):
            """Return the number of sets in the provided Set hand.
            Parameters:
            cards (list(str)) a list of twelve cards as 4-bit integers in
            base 3 as strings, such as ["1022", "1122", ..., "1020"].
            Returns:
            (int) The number of sets in the hand.
            Raises:
            ValueError: if the list does not contain a valid Set hand, meaning
            - there are not exactly 12 cards,
            - the cards are not all unique,
            - one or more cards does not have exactly 4 digits, or
            - one or more cards has a character other than 0, 1, or 2.
            """
            for i in cards:
                if len(i) != 4:
                    raise ValueError("Please enter a list of exactly 12 unique "+\
                                     "cards, each with 4 digits, each of which being either 0, 1, or
                for j in i:
                    if j not in str(list(range(3))):
                        raise ValueError("Please enter a list of exactly 12 unique "+\
                                         "cards, each with 4 digits, each of which being either 0, 1, or
            if len(cards) != 12 or len(set(cards)) != len(cards):
                raise ValueError("Please enter a list of exactly 12 unique "+\
                                 "cards, each with 4 digits, each of which being either 0, 1, or
            count = 0
            for i in list(combinations(cards,3)):
                if is_set(i[0],i[1],i[2]):
                    count+=1
            return count

        def is_set(a, b, c):
            """Determine if the cards a, b, and c constitute a set.
            Parameters:
            a, b, c (str): string representations of 4-bit integers in base 3.
            For example, "1022", "1122", and "1020" (which is not a set).
            Returns:
            True if a, b, and c form a set, meaning the ith digit of a, b,
            and c are either the same or all different for i=1,2,3,4.
            False if a, b, and c do not form a set.
            """
            for i in range(4):
                if len(set([a[i], b[i], c[i]])) not in [1,3]:
                    return False
```

```
            return True
```

Let's test our implementation using !py.test. It looks like it's a success!

```
In [56]: !py.test
```

```
============================= test session starts ==============================
platform darwin -- Python 3.6.5, pytest-3.6.2, py-1.5.3, pluggy-0.6.0
rootdir: /Users/hbeard/Documents/GitHub/BootCamp2018/ProbSets/Comp/ProbSet1, inifile:
plugins: remotedata-0.2.1, openfiles-0.3.0, doctestplus-0.1.3, cov-2.5.1, arraydiff-0.2
collected 34 items                                                             test_fraction.py
test_month_length.py ...                                                 [ 44%]
test_operate.py ...                                                      [ 64%]
test_setgame.py ...                                                      [ 82%]
test_smallest_factor.py ...                                              [100%]

========================== 34 passed in 0.22 seconds ===========================
```

```
In [54]: !py.test --cov
```

```
============================= test session starts ==============================
platform darwin -- Python 3.6.5, pytest-3.6.2, py-1.5.3, pluggy-0.6.0
rootdir: /Users/hbeard/Documents/GitHub/BootCamp2018/ProbSets/Comp/ProbSet1, inifile:
plugins: remotedata-0.2.1, openfiles-0.3.0, doctestplus-0.1.3, cov-2.5.1, arraydiff-0.2
collected 34 items                                                             test_fraction.py
test_month_length.py ...                                                 [ 44%]
test_operate.py ...                                                      [ 64%]
test_setgame.py ...                                                      [ 82%]
test_smallest_factor.py ...                                              [100%]

---------- coverage: platform darwin, python 3.6.5-final-0 -----------
Name                      Stmts   Miss  Cover
---------------------------------------------
fraction.py                  33      0   100%
month_length.py              10      0   100%
operate.py                   14      0   100%
setgame.py                   20      0   100%
smallest_factor.py            5      0   100%
test_fraction.py             60      0   100%
test_month_length.py         15      0   100%
test_operate.py              22      0   100%
test_setgame.py              65      0   100%
test_smallest_factor.py      13      0   100%
---------------------------------------------
TOTAL                       257      0   100%


========================== 34 passed in 0.18 seconds ===========================
```

```
In [55]: !py.test --cov-report html --cov

============================= test session starts ==============================
platform darwin -- Python 3.6.5, pytest-3.6.2, py-1.5.3, pluggy-0.6.0
rootdir: /Users/hbeard/Documents/GitHub/BootCamp2018/ProbSets/Comp/ProbSet1, inifile:
plugins: remotedata-0.2.1, openfiles-0.3.0, doctestplus-0.1.3, cov-2.5.1, arraydiff-0.2
collected 34 items                                                             test_fraction.py
test_month_length.py ...                                                 [ 44%]
test_operate.py ...                                                      [ 64%]
test_setgame.py ...                                                      [ 82%]
test_smallest_factor.py ...                                              [100%]

---------- coverage: platform darwin, python 3.6.5-final-0 -----------
Coverage HTML written to dir htmlcov


========================== 34 passed in 0.18 seconds ===========================
```

4 Object Oriented Programming

Problem 1

```
In [94]: class Backpack:
             """
             A backpack object class. Has a name, color, maximum size,
             and a list of contents.

             Attributes:
                 name (str): the name of the backpack's owner.
                 color (str): the color of the backpack.
                 max_size (int): the maximum amount of items in
                                 the 'contents' attribute.
                 contents (list): the contents of the backpack.
             """
             def __init__(self, name, color, max_size = 5):
                 """
                 Set the name and initialize and empty list of contents
                 of the backpack.

                 Params:
                     name (str): the name of the backpack's owner.
                     color (str): the color of the backpack.
                     max_size (int): the maximum amount of items in
                                     the 'contents' attribute.
                 """
                 self.name = name
                 self.color = color
```

```
        self.max_size = max_size
        self.contents = []

    def put(self, item):
        """
        Adds an 'item' to the backpack's list of contents by
        appending the 'contents' list by that item.
        """
        if len(self.contents) + 1 > self.max_size:
            print("No Room!")
        else:
            self.contents.append(item)

    def dump(self):
        """
        Empties the backpack entirely; resets 'contents'
        back to the empty list [].
        """
        self.contents = []

    def take(self, item):
        """
        Remove 'item' from the backpack's list of contents.
        """
        self.contents.remove(item)
```

Testing to see if it works:

```
In [97]: def test_backpack():
             testpack = Backpack("Barry","black")
             if testpack.name != "Barry":
                 print("Backpack.name assigned incorrectly")
             for item in ["pencil","pen","paper","computer"]:
                 testpack.put(item)
             print("Contents:",testpack.contents)
             print(testpack.max_size)
             testpack.put("binder")
             testpack.put("giraffe") # no room for one more item
             testpack.take("paper")
             print(testpack.contents)
             testpack.dump()
             print(testpack.contents)

In [98]: test_backpack()

Contents: ['pencil', 'pen', 'paper', 'computer']
5
No Room!
['pencil', 'pen', 'computer', 'binder']
```

[]

Problem 2

In [ ]:
```
class Jetpack (Backpack):
    """
    A jetpack subclass of Backpack.
    Has a name, color, max size, fuel, and a list of contents.

    Attributes:
        name (str): the name of the backpack's owner.
        color (str): the color of the backpack.
        max_size (int): the maximum amount of items in
                        the 'contents' attribute.
        fuel (int): amount of fuel in the jetpack (to be used
                    for flying)
        contents (list): the contents of the backpack.
    """
    def __init__(self, name, color, max_size = 2, fuel = 10):
        """
        Constructs a jetpack object by initializing name and color.
        Keyword arguments include max_size and fuel. Contents attribute
        is also initialized.

        Params:
            name (str): the name of the person who owns the jetpack.
            color (str): the color of the jetpack.
            max_size (int): the maximum number of contents in the jetpack.
            fuel (int): the amount of fuel in the jetpack.
        """
        self.name = name
        self.color = color
        self.max_size = max_size
        self.fuel = fuel
        self.contents = []
    def fly(self, fuel_burn):
        """
        Accepts an amount of fuel to be burned and decreases
        the fuel attribute by that amount.

        Params:
            fuel_burn (int): amount of fuel to be burned for the flight.
        """
        if fuel_burn > self.fuel:
            print("Not enough fuel!")
        else:
            self.fuel -= fuel_burn
```

```
        def dump(self):
            """
            Completely empites the contents and fuel attributes.
            """
            self.contents = []
            self.fuel = 0
```

Problem 3

```
In [9]: class Backpack:
            """
            A backpack object class. Has a name, color, maximum size,
            and a list of contents.

            Attributes:
                name (str): the name of the backpack's owner.
                color (str): the color of the backpack.
                max_size (int): the maximum amount of items in
                                the 'contents' attribute.
                contents (list): the contents of the backpack.
            """
            def __init__(self, name, color, max_size = 5):
                """
                Set the name and initialize and empty list of contents
                of the backpack.

                Params:
                    name (str): the name of the backpack's owner.
                    color (str): the color of the backpack.
                    max_size (int): the maximum amount of items in
                                    the 'contents' attribute.
                """
                self.name = name
                self.color = color
                self.max_size = max_size
                self.contents = []

            def put(self, item):
                """
                Adds an 'item' to the backpack's list of contents by
                appending the 'contents' list by that item.
                """
                if len(self.contents) + 1 > self.max_size:
                    print("No Room!")
                else:
                    self.contents.append(item)

            def dump(self):
```

```
        """
        Empties the backpack entirely; resets 'contents'
        back to the empty list [].
        """
        self.contents = []

    def take(self, item):
        """
        Remove 'item' from the backpack's list of contents.
        """
        self.contents.remove(item)
    def __eq__(self, other):
        """
        Returns True if and only if the name, color, and
        contents of another Backpack object (taken in as an
        argument) are equal to its own name, color, and contents.

        Params:
            other (Backpack object): another backpack that we're
                                     interested in comparing our
                                     own backpack's attributes to.
        """
        return self.name == other.name and \
               self.color == other.color and \
               self.contents == other.contents
    def __str__(self):
        """
        Returns a string summary of attributes about the
        specific instantiation of this Backpack.
        """
        s  = "Owner: \t\t"  + str(self.name) + "\n"
        s += "Color: \t\t"  + str(self.color) + "\n"
        s += "Size: \t\t"   + str(len(self.contents)) + "\n"
        s += "Max Size: \t" + str(self.max_size) + "\n"
        s += "Contents: \t" + str(self.contents)
        return s
```

Problem 4

```
In [79]: from math import sqrt

         class ComplexNumber:
             """
             ComplexNumber class. Contains real and imaginary parts, both floats or ints (unless
             Params: real (the real part), and imag (the imaginary part)
             """
             def __init__(self, real, imag):
                 """
```

```
        Initializes an object with real and imaginary parts, specified as arguments.
        """
        self.real = real
        self.imag = imag
    def conjugate(self):
        """
        Returns the complex conjugate of the number.
        """
        return ComplexNumber(self.real, -1 * self.imag)
    def __str__(self):
        """
        If we call print(z) for ComplexNumber z, we should get (Re(z)+/-Im(z)j) for Re(
        """
        if self.real == 0:
            return str(self.imag) + "j"
        elif self.imag >= 0:
            return "(" + str(self.real) + "+" + str(abs(self.imag)) + "j)"
        elif self.imag < 0:
            return "(" + str(self.real) + "-" + str(abs(self.imag)) + "j)"
    def __abs__(self):
        """
        Returns the 'norm' of the complex number, defined as \sqrt{ \Re(z)^2 + \Im(z)^2
        """
        return sqrt(self.real ** 2 + self.imag ** 2)
    def __eq__(self, other):
        """
        Defines an equals magic method. z1 == z2 iff Re(z1)==Re(z2) and Im(z1)==Im(z2).
        """
        return self.real == other.real and self.imag == other.imag
    def __add__(self, other):
        """
        Defines an addition magic method. Adds component-wise
        """
        return ComplexNumber(self.real + other.real, self.imag + other.imag)
    def __sub__(self, other):
        """
        Defines a subtraction magic method. Subtracts component-wise
        """
        return ComplexNumber(self.real - other.real, self.imag - other.imag)
    def __mul__(self, other):
        """
        FOIL: (a+bi)*(c+di) = ac + adi + bic + bidi
                            = ac + adi + bic - bd
                            = (ac - bd) + (ad + bc)i
                            = ComplexNumber(ac-bd, ad+bc)
        """
        re = self.real * other.real - self.imag * other.imag
        im = self.real * other.imag + self.imag * other.real
```

```
        return ComplexNumber(re, im)
    def __truediv__(self, other):
        """
        Multiply by conj:
            (a+bi)/(c+di) = (a+bi)/(c+di)*(c-di)/(c-di)
                          = ...
                          = ((ac+bd)/(c^2+d^2)) + ((bc-ad)/(c^2+d^2))i
        """
        n = self.__mul__(other.conjugate())
        d = other.__mul__(other.conjugate())
        return ComplexNumber(n.real / d.real, n.imag / d.real)
```

Below is the testing function:

```
In [111]: def test_ComplexNumber(a,b,c,d):
              """
              Tests the ComplexNumber class construction to see if all the defined methods work
              """
              success = True
              py_cnum, my_cnum = complex(a,b), ComplexNumber(a,b)
              py_cnum2, my_cnum2 = complex(c,d), ComplexNumber(c,d)

              # Validate the constructor.
              if my_cnum.real != a or my_cnum.imag != b:
                  print("__init__() set self.real and self.imag incorrectly")
                  success = False

              # Validate conjugate() by checking the new number's imag attribute.
              if py_cnum.conjugate().imag != my_cnum.conjugate().imag:
                  print("conjugate() failed for", py_cnum)
                  success = False

              # Validate __str__().
              if str(py_cnum) != str(my_cnum):
                  print("__str__() failed for", py_cnum)
                  success = False

              # Validate __abs__().
              if abs(py_cnum) != abs(my_cnum):
                  print("__abs__() failed for", py_cnum)
                  success = False

              # Validate __eq__().
              if (complex(a,b) == py_cnum) != (ComplexNumber(a,b) == my_cnum):
                  print("__eq__() failed for", py_cnum)
                  success = False

              # Validate __add__():
```

```
            if my_cnum + my_cnum2 != py_cnum + py_cnum2:
                print("__add__() failed for", py_cnum)
                success = False

            # Validate __sub__():
            if my_cnum - my_cnum2 != py_cnum - py_cnum2:
                print("__sub__() failed for", py_cnum)
                success = False

            # Validate __mul__():
            if my_cnum * my_cnum2 != py_cnum * py_cnum2:
                print("__mul__() failed for", py_cnum)
                success = False

            # Validate __truediv__():
            if my_cnum2.real == 0 and my_cnum2.imag == 0:
                pass # avoiding dividing by zero error
            else:
                my_quotient = my_cnum / my_cnum2
                py_quotient = py_cnum / py_cnum2
                if    (round(my_quotient.real,8) != round(py_quotient.real,8)) \
                   or (round(my_quotient.imag,8) != round(py_quotient.imag,8)) :
                    # rounded to avoid round-off error
                    print("__truediv__() failed for", py_cnum)
                    success = False

            return success

In [112]: successes = []
          for i in range(-5,6):
              for j in range(-5,6):
                  for k in range(-5,6):
                      for l in range(-5,6):
                          successes.append(test_ComplexNumber(i,j,k,l))
          if(all(successes) == True):
              print("It works!")
          else:
              print("It failed :(")

It works!
```

As we can see, it looks like it worked!

5 Exceptions and File I/O

Problem 1

```
In [141]: def arithmagic():
              """
              Prompts user for related sequences of 3-digit integers, eventually arriving at 108
              Tests each time to see if user enters valid entries. If not, we raise a ValueError
              """
              step_1 = input("Enter a 3-digit number where the first "
                          + "and last digits differ by 2 or more: ")
              if any(list(step_1)) not in list(range(0,10)) or len(str(step_1)) != 3:
                  raise ValueError("Must be a 3-digit integer.")
              if abs(int(str(step_1)[0]) - int(str(step_1)[-1])) < 2:
                  raise ValueError("First and last digits must differ "
                                   + "by at least 2.")

              step_2 = input("Enter the reverse of the first number, "
                          + "obtained by reading it backwards: ")
              if str(step_2)[::-1] != str(step_1):
                  raise ValueError("Must be the reverse of the first number.")

              step_3 = input("Enter the positive difference of these "
                          + "numbers: ")
              if int(step_3) != abs(int(step_1) - int(step_2)):
                  raise ValueError("Must be the positive difference of "
                                   + "the first two numbers.")

              step_4 = input("Enter the reverse of the previous result: ")
              if str(step_4)[::-1] != str(step_3):
                  raise ValueError("Must be the reverse of the third number.")

              print((str(step_3) + "+" + str(step_4) + "= 1089 (ta-da!)"))
```

Problem 2

```
In [151]: from random import choice
          def random_walk(max_iters=1e3):
              """
              Returns a random walk (int) for 1e3 max iterations.
              """
              walk=0
              directions=[1,-1]
              for i in range(int(max_iters)):
                  walk+=choice(directions)
              return walk
```

```
In [161]: from random import choice
          def random_walk(max_iters=1e10):
```

```
    """
    Returns a random walk (int) for 1e3 max iterations.
    Aborts and reports the iteration and walk amounts if ^C is entered by user before
    """
    iteration = 0
    try:
        walk=0
        directions=[1,-1]
        for i in range(int(max_iters)):
            walk+=choice(directions)
            iteration += 1
    except KeyboardInterrupt:
        print("Process interrupted at iteration " + str(iteration))
    else:
        print("Process completed")
    finally:
        return walk
```

Problem 3

```
In [99]: class ContentFilter:
             def __init__(self, file):
                 """
                 Reads context of text file, repeaitng until a valid name is entered, in a while
                 """
                 error = True
                 while error:
                     try:
                         with open(file,"r") as f:
                             pass
                     except:
                         file = input("Please enter a valid file name: ")
                     else:
                         error = False
                 with open(file,"r") as f:
                     self.name = f.name
                     self.contents = f.read()
                 print("File is closed: ", f.closed)
```

Let's put it to the test with some samples:

```
In [100]: c = ContentFilter("file.txt")
          print("Name: ",c.name)
          print("Contents:",c.contents)
```

```
File is closed:  True
Name:  file.txt
Contents: This is the file text.
```

```
THIS IS SOME MORE TEXT.
blah blah blah


still testing

This is the last line.
```

```
In [101]: c = ContentFilter("not-a-file.txt")
          print("Name: ",c.name)
          print("Contents:",c.contents)
```

```
Please enter a valid file name: still-not-a-file.txt
Please enter a valid file name: file.txt
File is closed:  True
Name:  file.txt
Contents: This is the file text.

THIS IS SOME MORE TEXT.
blah blah blah


still testing

This is the last line.
```

Problem 4

Putting it all together, we have the more expanded `ContentFilter` class:

```
In [314]: class ContentFilter:
              """
              Defines a ContentFilter class, with attributes name and contents (of the file), an
              the contents of the file. the only Param is the file name, assumed to be in the sa
              """
              def __init__(self, file):
                  """
                  Instantiates a ContentFilter object, attached to the file specified by the 'fi
                  """
                  error = True
                  while error:
                      try:
                          with open(file,"r") as f:
                              pass
                      except:
                          file = input("Please enter a valid file name: ")
                      else:
```

```
            error = False
        with open(file,"r") as f:
            self.name = f.name
            self.contents = f.read()

    def uniform(self, other_file, mode = "w", case = "upper"):
        """
        Copies all the text in the underlying file and converts to the same case, then
        to another file, specified in the argument. Defaults to uppercase.
        Can specify writing mode (writing, writing new, or appending). Defaults to wri
        """
        try:
            if mode not in ["w","x","a"]:
                raise ValueError("Please specify only 'w', 'x', or 'a'.")
            if case not in ["upper","lower"]:
                raise ValueError("Please specify only 'upper' or 'lower'")
        except ValueError as e:
            print("ValueError:",e)
        else:
            if case == "upper":
                with open(other_file, mode) as o_f:
                    o_f.write(self.contents.upper())
            elif case == "lower":
                with open(other_file, mode) as o_f:
                    o_f.write(self.contents.lower())

    def reverse(self, other_file, mode = "w", unit = "line"):
        """
        Copies all the text in the underlying file and reverses it, then writing
        to another file, specified in the argument. Can either reverse word-by-word wi
        or can reverse line-by-line with fixed words. Defaults to reversing line-by-li
        Can specify writing mode (writing, writing new, or appending). Defaults to wri
        """
        try:
            if mode not in ["w","x","a"]:
                raise ValueError("Please specify only 'w', 'x', or 'a'.")
            if unit not in ["line","word"]:
                raise ValueError("Please specify only 'line' or 'word'")
        except ValueError as e:
            print("ValueError:",e)
        else:
            if unit == "line":
                with open(other_file, mode) as o_f:
                    lines = []
                    for i in self.contents.split("\n")[::-1]:
                        lines.append(i + "\n")
                    o_f.writelines(lines)
            elif unit == "word":
```

```
            with open(other_file, mode) as o_f:
                for i in self.contents.split("\n"):
                    words = ""
                    for j in i.split(" ")[::-1]:
                        words += j + " "
                    o_f.write(words + "\n")

    def transpose(self, other_file, mode="w"):
        """
        Assuming an equal number of words per line.
        Copies all the text in the underlying file and "transposes" it (where each wor
        is a "cell" in matrix), then writing to another file, specified in the argumen
        Can specify writing mode (writing, writing new, or appending). Defaults to wri
        """
        try:
            if mode not in ["w","x","a"]:
                raise ValueError("Please specify only 'w', 'x', or 'a'.")
        except ValueError as e:
            print("ValueError:",e)
        else:
            rows = self.contents.split("\n")
            cols = []
            row = []
            for i in rows:
                row.append(i.split(" "))
            for i in range(len(rows)):
                rows[i] = row[i]
            for j in range(len(rows[0])):
                col = ""
                # assuming same number of words per line
                for i in rows:
                    col += i[j] + " "
                cols.append(col)

            transposed = ""

            for j in cols:
                transposed += j + "\n"

            with open(other_file, mode) as o_f:
                o_f.write(transposed)
```