Homework 8 Submission

Your Name:

In [156]:
```
# Helen Gao
```

In [157]:
```
# this is what the header would be if this was a .py file made in pycharm
# (not really sure if this is still needed since this is a notebook submission)
# (but here it is anyways)
# ass8.py
#
# write four functions
# the first function generates the elements of a collatz sequence from a given starting number until 1
# the second function generates the names of the months in sequence after a given month
# the third function takes a nested list and returns a flat list without nones or empty sublists
# the fourth function returns whether two strings are anagrams
# Usage:
#     % python ass8.py
#
# Helen Gao, 7-25-2018 by 11am
```

In [158]:
```python
# A generator that returns the unique elements of a Collatz sequence
# Takes an integer.  Returns a sequence of integers
# Given 10, it should return the elements 10, 5, 16, 8, 4, 2, 1
#
def collatz_gen(start):

    # returns the starting number as the first element
    yield start

    # since collatz needs to end on one this runs until the second to last element
    # (which is the last element used to generate new elements) is not one
    while start != 1:

        # if the number is even
        if not start % 2:

            # divide the number by 2 and return an integer using integer division
            start = start//2

        # if the number is odd
        else:

            # multiply the number by 3 and add 1
            start = 3*start+1

        # yield the next element
        # since this goes until the element used in the while loop is 2 the final element will be 1
        yield start


gen = collatz_gen(10)
try:
    while True:
        print(next(gen))
except StopIteration:
    print("That's all")
```

```
10
5
16
8
4
2
1
That's all
```

In [159]:
```
# A generator that returns the names of the months in sequence
# Take a string.  Returns sequence of strings
# Given May, it should return June, July, August, ...
#
def next_month(name):

    # creating a list called months of all the months in the year
    months = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'Octobe
r', 'November', 'December']

    # the variable month name is set to the given name
    month_name = name

    # while true means this runs indefinitely
    while True:

        # the month name becomes the next month name
        # one is added to the index of the given name so that the new number is the index of the next month
        # since the months repeat the function finds the remainder of the new number divided by 12
        # this is then indexed in the months list to find the next month
        month_name = months[(months.index(month_name)+1) % 12]

        # yield the next month name
        yield month_name


gen = next_month('May')

for i in range(20):
    print(next(gen))
```

```
June
July
August
September
October
November
December
January
February
March
April
May
June
July
August
September
October
November
December
January
```

In [160]:
```
# Take a nested list and return a flat list with no Nones nohow
# Given [1, [2,3,None,4], [], [None], 5] it should return [1,2,3,4,5]
#
def flatten(lst):

    # creating a list called flattened to hold the flat list
    flattened = []

    # for every item in the given list
    for item in lst:

        # if the item is a list
        # meaning the list is nested
        if type(item) == list:

            # for every value in the list item
            for val in item:

                # if the value is not none
                # this gets rid of the nones in the list
                if val is not None:

                    # add the value to the flat list
                    flattened.append(val)

        # if the item is not a list
        # nor is the item none
        # this gets rid of the nones
        elif item is not None:

            # add the value to the flat list
            flattened.append(item)

    # return the final flattened list
    return flattened


print(flatten([1, [2,3,None,4], [], [None], 5]))
```

```
[1, 2, 3, 4, 5]
```

In [161]:
```
# Are two words anagrams?
# Takes two strings, returns a Boolean
# Ignore case
#
def are_anagrams(word1, word2):

    # making the first string lowercase
    # this ignores the case
    word1 = word1.lower()

    # making the second string lowercase
    # this ignores the case
    word2 = word2.lower()

    # if the words are the same
    if word1 == word2:

        # since a word is not its own anagram the function returns false
        return False

    # if the inputs are not the same word
    else:

        # if the sorted version of word1 is equal to the sorted version of word2
        # if letters used in both words are the same then the words are anagrams
        if sorted(word1) == sorted(word2):

            # since the words are anagrams the function returns true
            return True

        # if the sorted words are different
        else:

            # since the words are not anagrams the function returns false
            return False


s1 = 'stop'
s2 = 'pots'
print(s1, s2, are_anagrams(s1, s2))

s1 = 'patter'
```

```
s2 = 'tapper'
print(s1, s2, are_anagrams(s1, s2))

s1 = 'mass'
s2 = 'last'
print(s1, s2, are_anagrams(s1, s2))
```

```
stop pots True
patter tapper False
mass last False
```

Unit Tests

In [162]:
```
import unittest

# A subclass of unittest.TestCase
class Ass8Test(unittest.TestCase):

    def test_collatz_5(self):
        gen = collatz_gen(5)
        nxt = next(gen)
        nxt = next(gen)
        nxt = next(gen)
        self.assertEqual(nxt, 8)

    def test_collatz_6(self):
        gen = collatz_gen(6)
        nxt = next(gen)
        nxt = next(gen)
        nxt = next(gen)
        self.assertEqual(nxt, 10)


    def test_next_month_May(self):
        gen = next_month('May')
        nxt = next(gen)
        self.assertEqual(nxt, 'June')

    def test_next_month_December(self):
        gen = next_month('December')
        nxt = next(gen)
        self.assertEqual(nxt, 'January')

    def test_next_month_January(self):
        gen = next_month('January')
        for i in range(12):
            nxt = next(gen)
        self.assertEqual(nxt, 'January')


    def test_no_matches_1(self):
        self.assertFalse(are_anagrams("diaper", "zombies"))

    def test_matches_1(self):
        self.assertTrue(are_anagrams("master", "stream"))
    def test_matches_2(self):
        self.assertTrue(are_anagrams("stream", "master"))

    def test_no_matches_3(self):
        self.assertFalse(are_anagrams("good", "goody"))

    def test_no_matches_4(self):
        self.assertFalse(are_anagrams("good", "good"))

    def test_matches_3(self):
        self.assertTrue(are_anagrams("Listen", "Inlets"))

    def test_matches_4(self):
        self.assertTrue(are_anagrams("Listen", "Inlets"))

    def test_matches_5(self):
        self.assertTrue(are_anagrams("regally", "largely"))

    def test_does_not_detect_non_anagrams_with_identical_checksum(self):
        self.assertFalse(are_anagrams("mass", "last"))

    def test_matches_5(self):
        self.assertTrue(are_anagrams("Carthorse", "Orchestra"))

    def test_anagrams_must_use_all_letters_exactly_once(self):
        self.assertFalse(are_anagrams("tapper", "patter"))

    def test_capital_word_is_not_own_anagram(self):
        self.assertFalse(are_anagrams("BANANA", "Banana"))

    def test_no_nesting(self):
        self.assertEqual(flatten([0, 1, 2]), [0, 1, 2])

    def test_flatten_integers(self):
        inputs = [1, [2, 3, 4, 5, 6, 7], 8]
        expected = [1, 2, 3, 4, 5, 6, 7, 8]
        self.assertEqual(flatten(inputs), expected)

    def test_two_level_nesting(self):
        inputs =   [0, 2, [2, 3], 8, 100, 4, [50], -2]
        expected = [0, 2,  2, 3,  8, 100, 4,  50, -2]
        self.assertEqual(flatten(inputs), expected)
    def test_two_level_None(self):
        inputs =   [0, 2, [2, None, 3], None, 8, 100, [None], 4, [50], -2]
        expected = [0, 2,  2,       3,        8, 100,         4,  50,  -2]
        self.assertEqual(flatten(inputs), expected)

    def test_all_values_are_none(self):
        inputs = [None, [None], None, None, [None, None], None, None]
        expected = []
        self.assertEqual(flatten(inputs), expected)

    # Additional tests for this track
    def test_empty_nested_lists(self):
        self.assertEqual(flatten([[]]), [])

    def test_strings(self):
        self.assertEqual(flatten(['0', ['1', '2']]), ['0', '1', '2'])


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

```
.......................
----------------------------------------------------------------------
Ran 23 tests in 0.023s

OK
```

```
Out[162]: <unittest.main.TestProgram at 0x1ff983a14a8>
```