Mod 2 Homework: Test-Driven Development

You should use test-driven development (TDD) on every assigment in this course from here on out. Relying on auto-graded tests in Gradescope will make these assignments tougher. **The first thing you should do on every assignment is write tests.** To encourage this, we will not make auto-tests available until a few days after this assignment goes live.

Note: We are modeling a card game traditionally called "SET!" (link) in this assignment. To avoid confusion with the python built-in type of the same name, we will refer to the game as "GROUP!"

GROUP! is a card game. Each card has some number of shapes on it. There are 4 attributes per card:

- number (1, 2, or 3)
- shape (diamond, squiggle, or oval)
- color (green, blue, or purple)
- shading (empty, striped, solid)

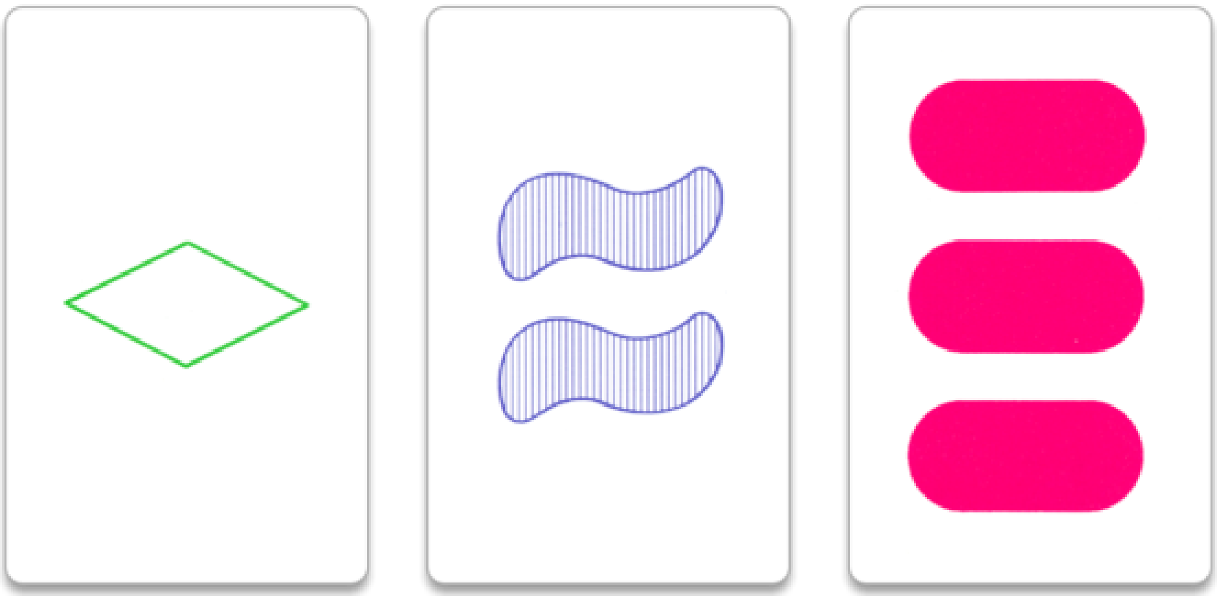

Figure 1: left to right - 1 open green diamond, 2 striped blue squiggles, 3 solid purple ovals.

In a game of GROUP!, cards are flipped up one at a time until someone spots a valid group. We'll cover what constitutes a "group" later. For now, we'll focus on building classes for GROUP! cards and decks.

`TestHw2.py`

When writing tests, we'll group all the unittests for a given class inside a single `unittest.TestCase` class:

```
class TestCard(unittest.TestCase):
   # all tests for the class Card will be here

class TestDeck(unittest.TestCase):
   # all tests for the class Deck will be here
```

Each `unittest.TestCase` class can contain multiple tests. We will use 1 test per piece of functionality:

```
class TestCard(unittest.TestCase):
   def test_init():
      """Tests that we can initialize cards w/ number/color/shading/shape"""

   def test_str():
      """test that we can get a good string representation of Card instances"""

   def test_eq(self):
      """Tests that two cards are equal iff all attributes are equal"""
```

Notice that each test includes a *docstring* - a string defined in triple-quotes on the first line of a method. We can access these docstrings with `help`:

```
>>> import TestHw2
>>> help(TestHw2.TestCard.test_init)
Help on function test_init in module TestHw2:

test_init()
    Tests that we can initialize cards w/ number/color/shading/shape
```

TODO 1: Implement unittests for class `Card`

- initialization (this test is written for you)
  - should be able to create cards with 4 attributes
  - should be able to access each attribute's value

  ```
  >>> c1 = Card(2, "green", "striped", "diamond")
  >>> c1.number
  2
  ```

- string representation
  - make sure we can use `str()` to get a good string representation

  ```
  >>> str(c1)
  Card(2, green, striped, diamond)
  ```

- equality
  - use magic method
  - takes 2 parameters: `self` and `other`
  - return `True` iff ("if and only if") all 4 attributes of both cards are equal

TODO 2: Implement functionality for class `Card`

Once you have written the tests above, you can start implementing the functionality for them in `hw2.py`. Continue until you pass all your tests, then move on.

TODO 3: Implement unittests for class `Deck`

In this section, start adding docstrings to each test as you write them. This is good practice to improve code readability.

- initialization
  - should create a deck with one copy of each possible card
  - By default, use the numbers/shapes/colors/shadings above
    - use lists for the default values, with the orders given above. The last card you should append to your list upon initialization should be 3 solid purple ovals.
  - allow users to specify their own numbers/shapes/colors/shadings, if desired
  - `len()` will be helpful for writing these tests - implement it using the length magic method, `__len__()`, in `Deck`.

```
>>> x = Deck()
>>> len(x) # by default, 3*3*3*3 = 81 cards
81
>>> my_nums = {1, 2}
>>> my_shapes = {"circles", "squares", "ovals"}
>>> my_cols = {'maroon', 'aqua', 'perywinkle', 'blue'}
>>> my_shadings = {'striped'}
>>> y = Deck(numbers=my_nums, shapes=my_shapes, colors=my_cols, shadings=my_shadings)
>>> len(y) # 2*3*4*1 = 24
24
```

  - Cards should be stored in a list. Treat the last item in the list as the top of the deck.
- `draw_top()`
  - draws and reveals (removes and returns) the top card in a deck
  - Remember, this is the last card in the list of cards representing your deck
  - if someone tries to `draw_top()` on an empty deck, raise an AttributeError. For testing errors, see the **Basic example** here.
- `shuffle()`
  - shuffles the deck (i.e. randomizes the order of the cards)
  - use `random.shuffle()` to do this
  - generally, `random.shuffle()` will give unpredictable (and thus untestable) results. You can work around this by fixing the random seed in your tests with `random.seed()`:

```
>>> L = [c for c in "abcde"]
>>> import random
>>> random.seed(652) # ensure "random" events always play out the same way
>>> random.shuffle(L)
>>> L
['a', 'e', 'c', 'd', 'b']
```

  - you will probably have to run this test at least once after implementing functionality with a fixed random seed to see which card will be on top.

TODO 4: Implement functionality for `class Deck`

Once you have written the tests above, implement the appropriate functionality in `hw2.py`.

TODO 5: Find groups

In GROUP!, cards are dealt from the top of the deck face up, one at a time. The goal is to be the first person to call out when a group appears. A "group" is any collection of three cards where, for *each* of the four attributes, either

- all cards share the same value (e.g. 3 cards with diamonds)
- all cards have different values (e.g. 1 card with diamonds, 1 with squiggles, and 1 with ovals)

Figure 1 gives a group where, for each of the 4 attributes, all cards have different values: 1 open green diamond vs 2 striped blue squiggles vs 3 solid purple diamonds.

Another valid group would be 2 open green diamonds, 2 open blue squiggles, and 2 open purple diamonds. Each card has *the same* number and shading (1 open . . .), and each card has *different* colors and shapes (. . . green/blue/purple diamonds/squiggles/diamonds).

- Write an algorithm `is_group()` that takes 3 cards as parameters, and returns a boolean denoting whether those cards are a valid group
  - Start by writing a unittest for this algorithm, including a docstring. Make sure that it returns `True` when expected *and* `False` otherwise, i.e. both of the following lines are necessary for a sufficient test:

```
self.assertTrue(is_group(c1, c2, c3))
self.assertFalse(is_group(c1, c2, c4))
```

Submitting

At a minimum, submit the following files:

- `hw2.py`
- `TestHw2.py`

Students must submit individually by the due date (typically Tuesday at 11:59 pm EST) to receive credit.

Grading

This homework will be partially manually graded. When manually grading, we will consider how well you demonstrate course objectives within a given assignment's constraints:

- Did you structure your code according to best practices in object-oriented programming?
- Did you thoroughly test your code using the unittest module?
- Did you choose and correctly implement the best data structures and algorithms?
- Is the code well-organized and documented? (use comments, whitespace, and reasonable naming conventions)

Feedback

If you have any feedback on this assignment, please leave it here.