# CS106AP Final Exam Solutions

Summer 2019

## 1. Trace

```
# Expected output:
[3, 4, 5]
[1, 2]
[3, 4, 5]
[]
[1, 2, 4]
[3, 4, 5]
[3, 4, 5]
[1, 2, 4]
```

***Common errors***

- Be careful with variable names here. Just because the parameter passed into **`foo()`** is renamed **`lst1`** inside **`foo()`**, it doesn't mean the it refers to the same object as the **`lst1`** variable inside **`bar()`**. The same is true for **`baz()`**.
- Since lists are mutable, this means that any changes made to the lists using **`.append()`** will persist across all of the functions (so the changes made in **`foo()`** will be seen inside **`bar()`**). However, as emphasized in class, reassigning a list variable will not change the original list (so although the **`.append()`** change made inside **`baz()`** will be seen inside **`bar()`**, setting the list to **`[]`** does not do anything to the lists inside **`bar()`**).

## 2. Counting Word Frequency

### Part A

```
def get_word_frequencies(filename, common_words):
     word_counts = {}
     with open(filename, ‘r’) as f:
          for line in f:
               words = line.split()
               for word in words:
                    word = word.lower()
                    if word not in common_words:
                         # Can alternatively use .get(word, 0) + 1
                         if word not in word_counts:
                              word_counts[word] = 0
                         word_counts[word] += 1
     return word_counts
```

### Part B

```
def get_n_most_frequent(word_counts, n):
     sorted_freqs = sorted(word_counts.items(),
                           key=lambda key_val:key_val[1],
                           reverse=True)
     return sorted_freqs[:n]
```

## 3. Structuring Article Data

### Part A

```
def get_genre_data(filenames):
    """
    Returns a dictionary containing article data, where the genre
    is the key, and a list of tuples containing article information
    is the value. Takes in a list of filenames, where each file
    corresponds to the article data for a single genre.

    Input:
        filenames (list of strings): list of files containing genre
                                     data
    Returns:
        genre_data (dict: string -> list of tuples): dict organized by
                                                     genre
    """
    data = {}

    for file in filenames:
        with open(file, 'r') as f:
            lines = f.readlines()
            genre = lines[0].strip()
            article_list = []
            for line in lines[1:]:
                article_name, author, num_views =
line.strip().split(',')
                article_list.append((article_name, author,
                                     int(num_views)))
            data[genre] = article_list
    return data
```

**Part B**

```
def get_author_data(genre_data):
    """
    From a dictionary structured with genres as the keys, returns
    a dictionary containing the same information, with the authors
    as the keys.

    Input:
        genre_data (dict: string -> list of tuples): dict organized by
                                                     genre

    Returns:
        author_data (dict: string -> list of tuples): dict organized
                                                      by author
    """
    author_data = {}

    for genre in genre_data:
        for article_info in genre_data[genre]:
            article_name, author, num_views = article_info
            new_article_info = (article_name, num_views, genre)
            if author not in author_data:
                author_data[author] = []
            author_data[author].append(new_article_info)

    return author_data
```

## 4. Dodger Game

**Full solution code for all parts**

```
from campy.graphics.gwindow import GWindow
from campy.graphics.gobjects import GOval, GRect
from campy.gui.events.mouse import onmouseclicked
from campy.gui.events.timer import pause
import random

WINDOW_WIDTH = 800
WINDOW_HEIGHT = 340
MIN_SPEED = 2.0
MAX_SPEED = 4.0

class DodgerGraphics():

    def __init__(self, window_width=WINDOW_WIDTH,
                       window_height=WINDOW_HEIGHT):
        """
        Initializes the class attributes for the DodgerGraphics
        class. This class should keep track of the GWindow, the
        the ball, and the square.
        """
        self.window = GWindow(width=window_width,
                              height=window_height)

        # Create and place a filled ball in the graphical window.
        self.ball = GOval(window_height / 2, window_height / 2)
        self.ball.filled = True
        self.vx = 0
        self.reset_ball()
        self.window.add(self.ball)

        # Create a filled square in the graphical window.
        square_x = window_width - (window_height / 2 - 1)
        square_y = 0
        self.square = GRect(window_height / 2 - 1,
                            window_height / 2 - 1,
                            x=square_x, y=square_y)
        self.square.filled = True
        self.window.add(self.square)

        # Initialize mouse listeners
        onmouseclicked(self.move_square)

    def reset_ball(self):
        """
        Resets the ball's x speed to a random speed and places
        the ball randomly in either the top left or bottom
        left corner.
        """
        self.vx = random.uniform(MIN_SPEED, MAX_SPEED)
        self.ball.x = 0
        if random.randint(0, 1):
            self.ball.y = self.window.height / 2
        else:
            self.ball.y = 0  # can omit since ball y defaults to 0

    def move_ball(self):
        """
        Moves the ball in the x direction (to the right).
        """
        self.ball.move(self.vx, 0)

    def clear_screen(self):
        """
        Removes all objects from the window.
        """
        self.window.remove(self.square)
        self.window.remove(self.ball)

    def ball_limit_reached(self):
        """
        Returns True if the ball's x has reached the
        x-value of the right side of the square.
        Returns False if the ball's x is greater than the
        x of the right side of the square.
        """
        return self.ball.x + self.ball.width >= self.window.width

    def collision_detected(self):
        """
        Returns True if the ball has collided with the
        square and False otherwise.
        """
        ball_top_right_x = self.ball.x + self.ball.width
        obj = self.window.get_object_at(ball_top_right_x, self.ball.y)
        return obj is not None

    def move_square(self, event):
        """
        If the click is in the top half of the window, moves
        the square to the top right corner of the window.
        If the click is in the bottom half of the screen, moves
        the square to the bottom right corner.
        """
        if event.y < self.window.height / 2:
            self.square.y = 0
        else:
            self.square.y = self.window.height / 2


#######

FRAME_RATE = 1000 / 120 # 120 frames per second.

def main():
    """
    Main gameplay loop. Every timestep should check if the
    ball
    """
    graphics = DodgerGraphics()

    while True:
        if graphics.collision_detected():
            graphics.clear_screen()
            break
        elif graphics.ball_limit_reached():
            graphics.reset_ball()
        graphics.move_ball()
        pause(FRAME_RATE)

if __name__ == '__main__':
    main()
```

## 5. Managing Newspaper Subscribers

```
SUBSCRIPTION_PRICE = 50

class PYTimesUser:
    def __init__(self, name, subscription_price=SUBSCRIPTION_PRICE):
        """
        Initialize all attributes.

        Input:
            name (string): name of PYTimesUser
            subscription_price (int): cost of newspaper subscription
        """
        self.name = name
        self.subscriptions = []
        self.subscription_price = subscription_price
        self.balance = 0

    def add_subscription(self, year):
        """
        Adds year to class subscription list.

        Input:
            year (int): year to add to subscription list
        """
        self.subscriptions.append(year)

    def subscribe(self, year):
        """
        Checks to see if PYTimesUser has enough money to purchase a
        subscription. If there is enough money, buys subscription for
        given year and adds the year to the subscription list. If
        there is not enough money, prints out an error message that
        includes the current balance (amount of money in account).

        Input:
            year (int): year for which to purchase a subscription
        """
        if self.balance >= self.subscription_price:
            self.add_subscription(year)
            self.balance -= self.subscription_price
        else:
            print("Your current balance is", self.balance)
    def deposit(self, amount):
        """
        Deposits money into the PYTimesUser's account.

        Input:
            amount (int): amount to deposit
        """
        self.balance += amount

    def buy_gift_subscription(self, other_user, year):
        """
        Attempts to purchase a gift subscription for another
        PYTimesUser. If PYTimesUser doesn't have enough money to
        purchase a subscription at the other user's subscription
        price, then it displays an error message and the given user's
        balance (amount of money in account). If the transaction is
        successful, the other user has a subscription for the given
        year, and it displays a success message including the names
        of both users.

        Input:
            other_user (PYTimesUser): user for whom to buy gift
                                      subscription
            year (int): year for which to purchase a subscription
        """
        if self.balance >= other_user.get_subscription_price():
            other_user.add_subscription(year)
            self.balance -= other_user.get_subscription_price()
            print(self.name, 'bought a subscription for',
                  other_user.get_name()).
        else:
            print("You have a current balance of", self.balance)

    def get_subscription_price(self):
        """Getter method for subscription price"""
        return self.subscription_price

    def get_name(self):
        """Getter method for name"""
        return self.name
```

## 6. One-Liners

### Part A

```
[fruit_info[0].upper() for fruit_info in fruits]
```

### Part B

```
[fruit_info[0] for fruit_info in fruits if fruit_info[1] < 0.5]
```

### Part C

```
max(fruits, key=lambda fruit_info: fruit_info[2])
```

### Part D

```
sorted(fruits, key=lambda fruit_info: fruit_info[1]*fruit_info[2],
       reverse=True)
```